Hes·so // VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Domaine Sciences de l'ingénieur
Rte du Rawyl 47
CH- 1950 Sion 2
Tél. +41 27 606 85 11
Fax +41 27 606 85 75
info@hevs.ch

www.hevs.ch

Filière Systèmes industriels

Orientation Infotronics

Diplôme 2012

Christopher Métrailler

Android docking station

Professeur	Dr Pierre-André Mudry
Expert	Prof. Fabien Vannel

Sion, le 9 juillet 2012

[1] *Ce rapport est l'original remis par l'étudiant*

[2] *L'école ne fournit aucune garantie quant à l'utilisation des résultats contenus dans ce rapport*

[3] *Ce document est la propriété de la HES-SO Valais, il ne peut être reproduit ou communiqué sans son autorisation écrite.*

Données du travail de diplôme
Daten der Diplomarbeit

☒ FSI ☐ FTV	Année académique / *Studienjahr* **2011/2012**	No TD / *Nr. DA* **it/2012/23**
Mandant / *Auftraggeber* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*	Etudiant / *Student* **Christopher Métrailler** Professeur / *Dozent* **Pierre-André Mudry**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein	Expert / *Experte (données complètes)* **Prof. Fabien Vannel** Hepia | Rue de la Prairie 4 | 1202 Genève	

Titre / *Titel*

Android docking station

Description et Objectifs / *Beschreibung und Ziele*

De plus en plus de téléphones Android sont présents sur le marché. En parallèle, de nouveaux accessoires voient le jour. Ces téléphones sont tous équipés d'un port USB qui permet de les configurer et de les utiliser via un PC pour y transférer des données.

L'objectif du projet est de concevoir un système embarqué communicant en USB avec le téléphone et permettant d'être utilisé comme « docking station ». Cette station sera compatible avec la majeure partie des téléphones Android et pourra être utilisée pour différentes applications comme une centrale météo, un lecteur audio amplifié, un réveil, etc. Le système embarqué réalisé devra être suffisamment modulaire pour pouvoir supporter l'ajout de différents types de capteur (température, hygrométrie, GPIO, autre...) pouvant être accédés depuis Android.

Les objectifs sont :

— Test de la carte électronique modulaire développée durant le projet de semestre
— Développement du logiciel générique Android d'intégration des périphériques
— Permettre l'accès à quelques périphériques Android du côté de la docking station
— Démonstrateur « proof of concept » permettant de démontrer le couplage entre la docking station et le système Android.

Délais / *Termine*

Attribution du thème / *Ausgabe des Auftrags:*
14.05.2012

Exposition publique / *Ausstellung Diplomarbeiten:*
31.08.2012

Remise du rapport / *Abgabe des Schlussberichts:*
09.07.2012 | 12h00

Défense orale / *Mündliche Verteidigung:*
Semaine / *Woche 36*

Signature ou visa / *Unterschrift oder Visum*

Responsable de l'orientation
Leiter der Vertiefungsrichtung: .....

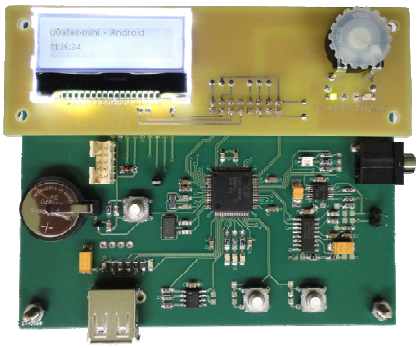
[1] Etudiant/*Student*: Métrailler Ch.

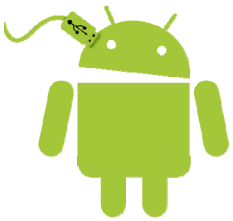
[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive et le caractère confidentiel du travail de diplôme qui lui est confié et des informations mises à sa disposition.*
Durch seine Unterschrift verpflichtet sich der Student, die Richtlinie einzuhalten sowie die Vertraulichkeit der Diplomarbeit und der dafür zur Verfügung gestellten Informationen zu wahren.

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats* .....

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Android docking station





Travail de diplôme
| édition 2012 |

Filière
Systèmes industriels

Domaine d'application
Infotronics

Professeur responsable
Dr. Pierre-André Mudry
pierre-andre.mudry@hevs.ch

Diplômant Christopher Métrailler

Objectif du projet

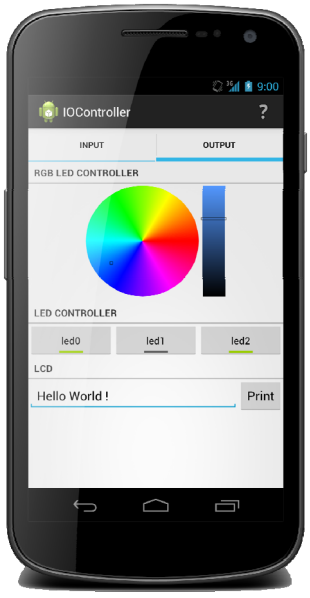
L'objectif du projet est de concevoir un système embarqué communicant en USB, utilisé comme docking station. Cette station est compatible avec la majeure partie des téléphones Android et peut être utilisée pour différentes applications, comme une centrale météo, un lecteur audio amplifié, un réveil, etc.

Méthodes | Expériences | Résultats

De plus en plus de téléphones Android sont présents sur le marché. Pour interagir avec ces derniers, une *docking station* modulaire a tout d'abord été développée. Elle dispose de plusieurs entrées/sorties, comme un écran, des leds ou encore des boutons qui peuvent être commandés via un téléphone Android. Elle fait également office de station multimédia permettant la diffusion sur des haut-parleurs de la musique stockée sur le téléphone.

Pour mettre en avant le système développé, trois applications de démonstration ont été écrites sur Android. Elles permettent de contrôler les entrées/sorties de la station d'accueil, d'utiliser le téléphone comme générateur de signaux et finalement de diffuser sa propre musique.

Au final, ce projet a permis de démontrer un couplage USB réussi entre la docking station et le système Android.



Contrôle des entrées/sorties

Générateur de signaux

HES-SO Valais
Route du Rawyl 47
1950 Sion

Tél. 027 606 85 11
URL www.hevs.ch

[Android/USB graphic by Roman Nurik.]

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

De plus en plus de téléphones Android™ sont présents sur le marché. En parallèle, de nouveaux accessoires voient le jour. Ces téléphones sont tous équipés d'un port USB qui permet de les configurer et de les utiliser via un PC pour y transférer des données par exemple. L'idée de ce projet est de concevoir un système embarqué communiquant en USB avec le téléphone.

Un grand nombre d'applications sont envisageables, dont voici quelques exemples :

- docking station pour téléphones Android (exemples : centrale météo, lecteur audio/réveil),
- commande et contrôle de périphériques via Android,
- cartes d'extension pour téléphones (entrées/sorties, lecteur NFC, adaptateur réseau).

La *docking station* modulaire développée sera compatible avec la majeure partie des téléphones Android et pourra remplacer un PC pour différentes applications. Elle disposera de plusieurs entrées/sorties, comme un écran, des LEDs ou encore des boutons qui pourront être commandés via un téléphone Android. Elle pourra également être utilisée comme station multimédia et permettra de diffuser du son sur des haut-parleurs. Enfin, des données pourront être transmises ou reçues sur le téléphone via différentes applications de démonstration écrites sur Android.

1.1 Android

Android [1] est un système d'exploitation *open source* (disponible via une licence Apache version 2) développé à la base pour les smartphones et les mobiles. Android, racheté par Google en 2005, est fondé sur un noyau Linux. De plus en plus d'appareils utilisent Android comme système d'exploitation, tels que tablettes, Netbook, téléviseurs, appareils électroménagers, montres, etc. Les applications Android sont principalement développées en Java et fonctionnent dans une machine virtuelle. Android intègre nativement tous les services proposés par Google, tels que Gmail, Google Maps, YouTube, etc. pour ne citer que les principaux.

Du fait de sa gratuité, de son code open source, ce système d'exploitation mobile évolue très rapidement, au point que plusieurs versions majeures sont publiées durant la même année. Il séduit un grand nombre de fabricants de téléphones mobiles et de développeurs, augmentant ainsi ses parts de marché. Aux États-Unis, le nombre de téléphones fonctionnant sur Android a dépassé celui d'Apple au premier trimestre 2010. En Suisse, on compte 36% de parts de marché[5] pour les smartphones Android. A titre d'information, Google a annoncé que 850'000 terminaux Android sont activés par jour (tous types d'appareils confondus).

1. Android is a trademark of Google Inc.

Versions

La rapide évolution d'Android implique un grand nombre de versions disponibles. La première version d'Android a été publiée le 23 septembre 2008. Toutes les versions ont un nom de code qui suit un ordre alphabétique basé sur le nom d'un dessert, comme le montre le tableau ci-dessous. Les versions sont classées de la plus ancienne à la plus récente.

Plateforme	Code	API level	Parts de marché
Android 1.5	Cupcake	3	0.3%
Android 1.6	Donut	4	0.7%
Android 2.1	Eclair	7	6.0%
Android 2.2	Froyo	8	23.1%
Android 2.3 - Android 2.3.2	Gingerbread	9	0.5%
Android 2.3.3 - Android 2.3.7	Gingerbread	10	63.2%
Android 3.0	Honeycomb	11	0.1%
Android 3.1	Honeycomb	12	1.0%
Android 3.2	Honeycomb	13	2.2%
Android 4.0 - Android 4.0.2	Ice Cream Sandwich	14	0.5%
Android 4.0.3	Ice Cream Sandwich	15	2.4%

Table 1.1 – Parts de marché des versions d'Android (avril 2012)

La répartition des parts de marché ainsi que l'historique des versions sont disponibles en annexe A. Google met périodiquement à jour ces chiffres[7]. Ces derniers ont été récoltés durant 14 jours, jusqu'au 2 avril 2012.

Les parts de marché d'Android 4 ont considérablement augmenté en quelques mois et représentent désormais plus de 10% des parts de marché. Les versions plus anciennes d'Android (2.2 et inférieures) tendent à disparaître et ne représentent plus qu'un téléphone sur cinq. Les logos des différentes versions d'Android sont illustrés ci-dessous :

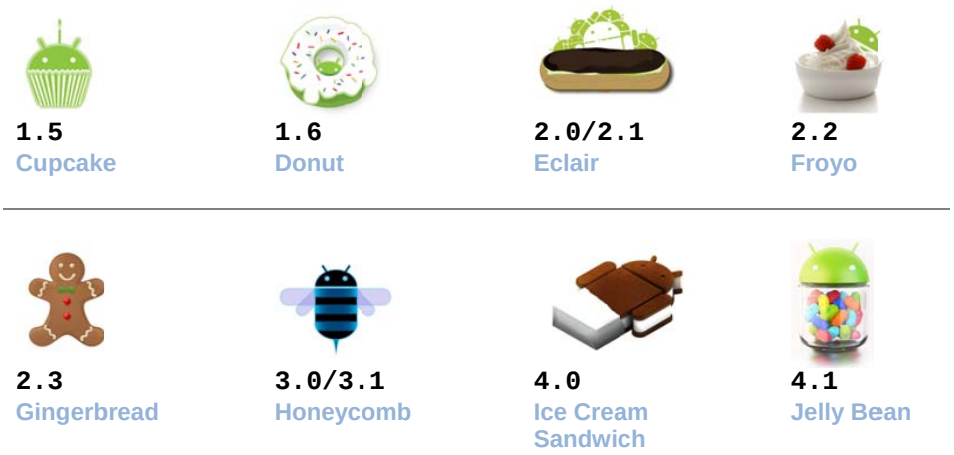


Figure 1.1 – Logos des versions d'Android

Les versions ICS (*Ice Cream Sandwich*, Android 4) sont les premières versions basées sur le noyau Linux en version 3. La dernière version d'Android 4.1 *Jelly Bean* a été annoncé pas Google et sera disponible au courant du mois de juillet.

Chapitre 2

Spécifications

Ce projet a été accepté suite à ma proposition, déposée auprès de l'unité *Infotronics* de l'école. Cette dernière est disponible en annexe B.

2.1 Cahier des charges

De plus en plus de téléphones Android sont présents sur le marché. En parallèle, de nouveaux accessoires voient le jour. Ces téléphones sont tous équipés d'un port USB qui permet de les configurer et de les utiliser via un PC pour y transférer des données. L'idée du projet est de concevoir un système embarqué communicant en USB avec le téléphone. Cette docking station sera compatible avec la majeure partie des téléphones Android et pourra remplacer un PC dans différentes applications (centrale météo, lecteur audio, réveil, contrôle de périphérique, ...). Le système embarqué réalisé devra être suffisamment modulaire pour pouvoir ajouter différents types de capteur (température, hygrométrie, GPIO, autre...).

Les objectifs du projet sont les suivants :

- Etude et documentation des contraintes hardware et software. Review des solutions existantes sur le marché.
- Réalisation d'un système électronique modulaire, potentiellement basé sur la carte *uGates*.
- Développement du logiciel générique Android d'intégration des périphériques.
- Permettre l'accès à quelques périphériques Android du côté de la docking station.
- Démonstrateur « proof-of-concept » permettant de démontrer le couplage entre la docking station et un système Android.

La description et les objectifs du projet sont tirés de la donnée du travail de diplôme, disponible en annexe C.

Compléments

Le couplage avec le système Android sera réalisé via une liaison USB. Dans un premier temps, la carte de développement *uGates* déjà existante sera utilisée. Une carte électronique dédiée sera développée dans un deuxième temps. La docking station restera au stade de prototype, le but étant de faire « la preuve du concept » à l'aide d'applications de démonstration Android. L'aspect mécanique de la docking station ne sera pas abordé dans ce projet.

2.2 Solutions existantes sur le marché

Avant de débuter le développement du projet, une petite analyse de marché a été réalisée. Elle a comme premier but de se documenter sur les produits similaires disponibles sur le marché ainsi que sur les technologies utilisées. Il faut tout d'abord différencier deux types de « docking station » disponibles sur le marché. Les stations d'accueil compatible avec iOS de type « Made for iPod, iPhone, iPad ® [1] » et celles compatibles avec les systèmes Android.

La première constatation est que les produits compatibles Apple sont très répandus et disponibles en grande quantité dans les magasins d'électronique. Ces stations d'accueil proposent des fonctions basiques, tels que horloge, réveil et bien sûr un amplificateur audio. Le connecteur propriétaire d'Apple [2], disponible sur tous les iPod, iPhone et iPad, facilite grandement le travail des fabricants puisque la sortie audio analogique stéréo est directement accessible et n'a plus qu'à être amplifiée. Des commandes séries normées peuvent être envoyées en utilisant « *Apple Accessory Protocol* » pour pouvoir par exemple passer à la chanson suivante, régler le volume, etc.

Les stations d'accueil grand public pour les systèmes Android sont pour l'heure peu répandues. En effet, un seul grand fabricant a choisi de se lancer dans ce domaine : Philips[3]. Plus d'une vingtaine de produits sont disponibles pour les produits Apple, contre trois modèles pour Android. Ces trois modèles (voir figure ci-dessous) intègrent des haut-parleurs (de puissance variable) et les fonctions d'horloge et de réveil. Un système mécanique complexe permet d'utiliser les stations d'accueil avec différents types et tailles d'appareils Android. Ils peuvent être rechargés via le port micro-USB de la station. La transmission audio est quant à elle réalisée via une liaison Bluetooth et non pas par USB.

AS111 *AS351* *AS851*

TABLE 2.1 – Stations d'accueil Philips pour Android

En conclusion, aucun produit grand public pour les systèmes Android n'utilise la liaison USB pour communiquer avec les téléphones. Le port USB est utilisé uniquement pour les recharger.

Kits de développement

Le système Android étant « open source », les kits de développement Android sont cette fois-ci plus répandus que les kits d'Apple. On peut citer trois catégories de kits qui communiquent avec les systèmes Android via USB. Ils se différencient principalement par le matériel qu'ils utilisent :

- Deux kits de développement PIC sont disponibles chez Microchip. *DM240415* et *DM320412* utilisent respectivement un PIC24F et un PIC32. Les kits de développement disposent d'un connecteur USB-A ainsi que d'entrées/sorties basiques telles que LEDs et boutons. Les PIC disposent d'un contrôleur USB Host/OTG intégré. Les kits Microchip ne sont pas spécifiquement développés pour les systèmes Android.
- Un autre kit est basé sur PIC24F. Le projet open source est nommé *IOIO*[15]. Il a été conçu pour communiquer spécifiquement avec des systèmes Android. Différents protocoles de communication sont implémentés. Une communication via Bluetooth est aussi disponible.

1. Apple MFi Program - https ://developer.apple.com/programs/mfi/
2. Pinout du connecteur (non officiel) - http ://www.allpinouts.org/index.php/Apple_iPod,_iPad_and_iPhone_dock.

– Un grand nombre de kits *Arduino[1]* sont disponibles. Ils se basent tous sur le même matériel. Ils utilisent le processeur *ATmega2560* et un contrôleur USB Host externe, le *MAX3421e*. Certains kits ont été spécialement développés pour Android. Ils sont disponibles sur des sites de vente d'électronique online à un prix très intéressant.

On peut encore citer d'autres kits « plus génériques », comme le kit *mbed[10]*. Il dispose d'un Cortex M3 (*NXP LPC1768*) avec contrôleur USB Host/OTG intégré. Le kit est minimaliste. La connectique USB ainsi que des E/S doivent être ajoutées en externe. Le processeur de ce kit est identique à celui utilisé sur la carte de développement *uGates* de l'école. Plusieurs types de microcontrôleurs peuvent être choisis. En choisissant un Cortex M3, des projets en C/C++ peuvent être développés. L'utilisation d'un contrôleur USB intégré est préférable car les vitesses de transfert sont plus rapides, la carte électronique est simplifiée et un composant externe peut être « économisé ».

Google a aussi présenté deux kits, basés sur la plateforme Arduino. Ces kits ne sont pas vendus, mais toutes les sources sont disponibles*[12]*. Voici le dernier kit présenté par Google en juillet 2012 :

FIGURE 2.1 – Kit 2012 de Google (source: engadget.com)

2.3 Matériel à disposition

Hardware

La carte de développement *uGates* a été développée en interne par l'école et est utilisée pour des projets de tous types. Elle se présente sous cette forme (PCB format europe 100x160mm) :

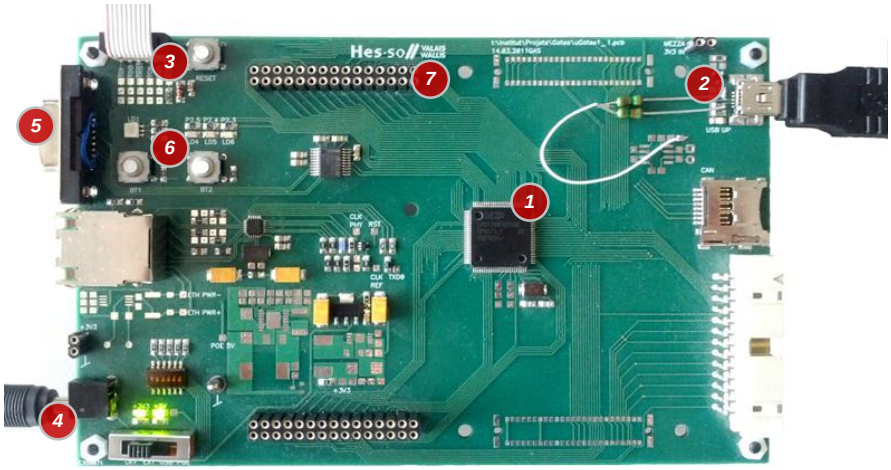


FIGURE 2.2 – Carte de développement *uGates*

Les caractéristiques de la carte sont les suivantes :

1. Microcontrôleur Cortex M3 de NXP (boîtier TQFP100). Le *LPC1768* intègre un contrôleur USB Host/Device/OTG et fonctionne à 80MHz.
2. Connecteur de type mini-USB B avec adaptateur externe vers USB-A.
3. Connecteur 10 pins pour JTAG avec un bouton reset.
4. Connecteur pour l'alimentation 5V du kit et du périphérique USB. Le switch permet de sélectionner la source d'alimentation.
5. Connecteur RS232. Liaison de debug avec un PC.
6. Deux boutons et trois LEDs sont disponibles ainsi qu'une LED RGB. Seule la couleur rouge est utilisable (défaut du PCB).
7. Plusieurs connecteurs (2.54mm et SMD) permettent de connecter des cartes mezzanines et d'accéder à toutes les E/S du microcontrôleur.

Toolchain

Une petite partie du projet a été consacrée à la mise en place des outils de développement. La mise en place du SDK d'Android est automatisée et n'a posé aucun problème. Toute la documentation est disponible sur le site officiel d'Android[6].

Pour la programmation du Cortex M3, deux solutions ont été envisagées :

– Utilisation d'un environnement de développement intégré, comme Red Suite, CrossWorks, IAR, etc. qui disposent des outils de programmation et de débogage. CooCox CoIDE est une bonne alternative et est gratuit.
– Utilisation d'un environnement de développement Open Source basé sur Eclipse, utilisant Sourcery G++, GCC et OpenOCD.

Les deux alternatives ont été testées. L'utilisation d'un environnement open source et cross-platform a été privilégié, bien que les outils soient plus difficiles à mettre en place. La procédure d'installation des toolchain pour la programmation et le débogage de la carte *uGates* est disponible en annexe D. Il est possible de compiler à la fois des projets C ou C++. Un plugin Doxygen a aussi été installé et permet de générer la documentation du code directement dans Eclipse.

Modifications de la carte *uGates*

Le microcontrôleur LPC1768 de NXP soudé sur la carte *uGates* dispose des fonctionnalités nécessaires pour le projet. Cependant, quelques modifications doivent être apportées sur la carte pour notamment supporter l'USB Host. Les modifications apportées sont provisoires et ne doivent pas endommager le kit. Ce dernier a été initialement prévu pour utiliser l'USB Device. Malheureusement, seul un connecteur de type mini-USB B est disponible.

Le schéma de connexion pour l'USB Host varie par rapport à l'USB Device. Des résistances de pull-down de 15k Ohm doivent être ajoutées sur le bus de données USB. Il est aussi nécessaire d'alimenter le périphérique USB directement avec l'alimentation 5V de la carte. Afin de connecter un périphérique USB Device avec un câble USB standard, un adaptateur mini USB-B vers USB-A a été utilisé. Cette solution permet d'éviter de modifier le PCB. Les modifications apportées et l'adaptateur USB sont visibles sur la figure 2.2.

Chapitre 3

Universal Serial Bus

3.1 Théorie

L'*USB[19]* est une norme qui décrit un protocole de transmission série dans lequel on retrouve un unique master appelé *Host*. Un nombre de périphériques (jusqu'à 127) peut s'y connecter à chaud (Plug and Play). Ce sont des *Device*, tels que clés USB, claviers/souris, imprimantes, etc. Dans le monde informatique, un ordinateur (PC, laptop) joue souvent le rôle de *Host*. En tant que master, c'est uniquement lui qui contrôle les données qui transitent sur le bus et qui dialogue avec les différents périphériques connectés. Il fournit aussi l'alimentation pour les périphériques, si ces derniers ne sont pas auto-alimentés. Dans sa version 2, l'USB permet d'atteindre des débits de l'ordre de plusieurs dizaines de Mo/s.
La norme USB 2.0[14] est très dense (plus de 650 pages). Elle décrit en détail son origine, les termes techniques utilisés, les types de connecteurs (partie mécanique), les flux de données, etc. Les différentes Classes USB standards, telle que la Classe HID, ne sont pas inclues dans ces pages. Le but de ce rapport n'est pas d'expliquer la norme USB en détail. Seuls des points importants utiles aux développements du projet seront décrits dans les prochains chapitres.

3.2 Appareils Android

Tous les téléphones et autres appareils fonctionnant sur Android sont équipés d'un port USB. Qu'il soit de type Mini-B ou Mirco-B, il est tout à fait indispensable. Il est utilisé pour recharger le téléphone ainsi que pour transférer des données depuis ou vers le téléphone avec un ordinateur par exemple. C'est aussi l'interface utilisée par les développeurs pour l'installation et le débogage d'applications.

Sous Windows, le « Gestionnaire de périphériques » permet de visualiser entre autres les appareils USB connectés, mais par la suite il n'est pas évident de visualiser toutes les informations propres au périphérique. C'est pourquoi le logiciel USBlyzer [1] a été utilisé. Il est spécialisé dans l'analyse USB et peut être assimilé à un « Wireshark pour USB ». Dans sa version de démonstration, l'analyseur USB n'est pas disponible.

USBlyzer donne une vue en arbre des périphériques USB connectés, comme le montrent les images ci-dessous :

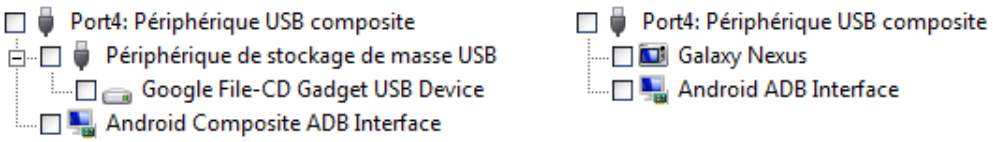


FIGURE 3.1 – Périphérique USB: Nexus S - Galaxy Nexus

1. USB Protocol Analyzer and USB Traffic Sniffer - http ://www.usblyzer.com/

Deux téléphones Android, le Nexus S à gauche et le Galaxy Nexus à droite ont été connectés. Tous les deux sont reconnus soit comme « Périphérique de stockage de masse » ou en tant que MPT [2].

Une deuxième interface est disponible sur tous les appareils Android, émulateur compris. C'est l'interface *Android Debug Bridge* (ADB). Utilisable en ligne de commande, elle permet d'installer des applications, de rediriger des ports TCP, de transférer des fichiers ou encore d'obtenir la console de débogage directement intégrée dans les outils de développement.

Contrairement au gestionnaire Windows, USBlyzer permet de visualiser simplement toutes les informations concernant chaque périphérique USB, le Samsung Nexus S dans notre cas. Un extrait des informations disponibles est présenté ci-dessous :

Device Descriptor Nexus

Offset	Field	Size	Value	Description
0	bLength	1	12h	
1	bDescriptorType	1	01h	Device
2	bcdUSB	2	0200h	USB Spec 2.0
4	bDeviceClass	1	00h	Class info in Ifc Descriptors
5	bDeviceSubClass	1	00h	
6	bDeviceProtocol	1	00h	
7	bMaxPacketSize0	1	40h	64 bytes
8	idVendor	2	18D1h	
10	idProduct	2	4E22h	
12	bcdDevice	2	0231h	2.31
14	iManufacturer	1	02h	"samsung"
15	iProduct	1	03h	"Nexus"
16	iSerialNumber	1	04h	"3132A344850900EC"
17	bNumConfigurations	1	01h	

Configuration Descriptor 1 Self Powered

Offset	Field	Size	Value	Description
0	bLength	1	09h	
1	bDescriptorType	1	02h	Configuration
2	wTotalLength	2	0037h	
4	bNumInterfaces	1	02h	
5	bConfigurationValue	1	01h	
6	iConfiguration	1	00h	
7	bmAttributes	1	C0h	Self Powered
	4..0: Reserved		...00000	
	5: Remote Wakeup		..0.....	No
	6: Self Powered		.1......	Yes
	7: Reserved (set to one) (bus-powered for 1.0)		1.......	
8	bMaxPower	1	FAh	500 mA

FIGURE 3.2 – Configuration USB Nexus S

A l'aide des informations fournies par le programme, il est possible d'en tirer le diagramme hiérarchique ci-dessous. Une seule configuration est disponible pour ce téléphone. On retrouve les deux interfaces décrites précédemment ainsi que les Endpoints d'entrées/sorties qui permettent de respectivement recevoir et envoyer des données sur cette interface.

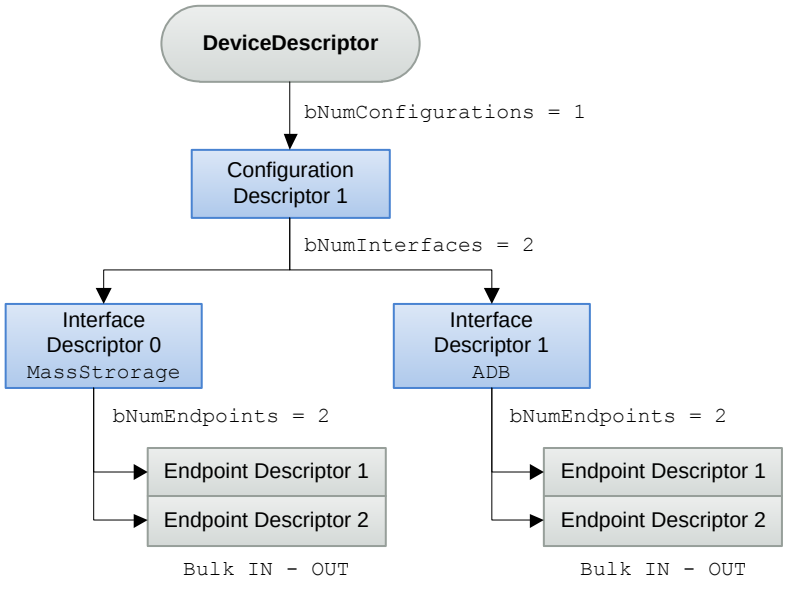


FIGURE 3.3 – *DeviceDescriptor* USB du Nexus S

La structure et les champs de chaque descripteur sont normés. Toutes les informations sont disponibles dans la spécification USB. Le site Beyondlogic[11] fournit en ligne et de manière simplifiée les informations importantes de la norme, dont celles des descripteurs USB.

2. Media Transfer Protocol - http ://en.wikipedia.org/wiki/Media_Transfer_Protocol

Chapitre 4

Android Open Accessory

4.1 Introduction

Jusqu'à présent, le port USB disponible sur tous les téléphones Android restait curieusement inaccessible aux développeurs, de manière officielle du moins.

L'apparition récente de tablettes Android fut le premier pas. Les versions Android *Honeycomb* sont dédiées et optimisées pour les tablettes. Pour une utilisation plus optimale de ces tablettes, comme la *Motorola XOOM* par exemple, le support de l'USB Host a été ajouté dès la version d'Android 3.1 et supérieures. Il est dès lors possible d'y connecter son appareil photo numérique pour transférer directement des images sur la tablette. Selon les pourcentages de répartition des versions, la part de marché des tablettes compatibles représente un peu plus de 3.3%, ce qui reste un marché faible et limité aux tablettes (voir tableau 1.1 page 9).

Afin d'étendre ce support, c'est récemment que Google annonça la nouvelle API *Android Open Accessory* [12], annonce faite lors de la conférence annuelle [1] de Google. La grande nouveauté de cette API est qu'il permet de connecter des « accessoires USB » avec les tablettes, mais aussi avec les téléphones qui ne supportent pas l'USB Host. Les *USB Accessories*, nommés accessoires, sont des cartes électroniques spécialement développées pour communiquer avec des périphériques Android en utilisant le protocole Android Accessory.

Le support de ce protocole a été introduit dans Android 2.3.4 (API version 10). Cette version est une mise à jour de la version d'Android 2.3.3. La version d'API n'a pas changé, mais cette mise à jour intègre l'API *Open Accessory* qui peut être inclue ou non par les fabricants de téléphones. L'image et le slogan ci-dessous proviennent de l'annonce de Google faite sur le blog officiel d'Android[13].

FIGURE 4.1 – A Bright Idea: Android Open Accessories

1. Conférence annuelle Google I/O 2011 réservée aux développeurs qui présente les derniers développements de la société (10 et11 mai 2011, San Fransisco). http ://www.google.com/events/io/2011/

4.2 Modes USB avec Android

Les appareils Android supportent une grande variété de périphériques et d'accessoires USB. Il faut distinguer deux modes de communication :

1. USB host : Android est l'USB Host. On peut y connecter des appareils génériques disponibles sur le marché tels que clés USB, appareils photo, claviers/souris, etc. Support natif pour Android 3 et Android 4 ICS.
2. USB accessory : le téléphone fonctionne en Device et est connecté à l'accessoire USB master. Ils doivent supporter un protocole de communication spécifique (*Android Accessory protocol*). Le mode USB Host n'est pas nécessaire sur Android ce qui augmente le nombre d'appareils compatibles. Support natif pour les versions d'Android 3.1 et plus récentes. Une librairie additionnelle doit être utilisée pour Android 2.3.4 (certains fabricants de téléphones peuvent désactiver son support).

Le schéma ci-dessous résume les caractéristiques de ces deux modes. Le master du bus USB se trouve à gauche et fournit l'alimentation au périphérique connecté.

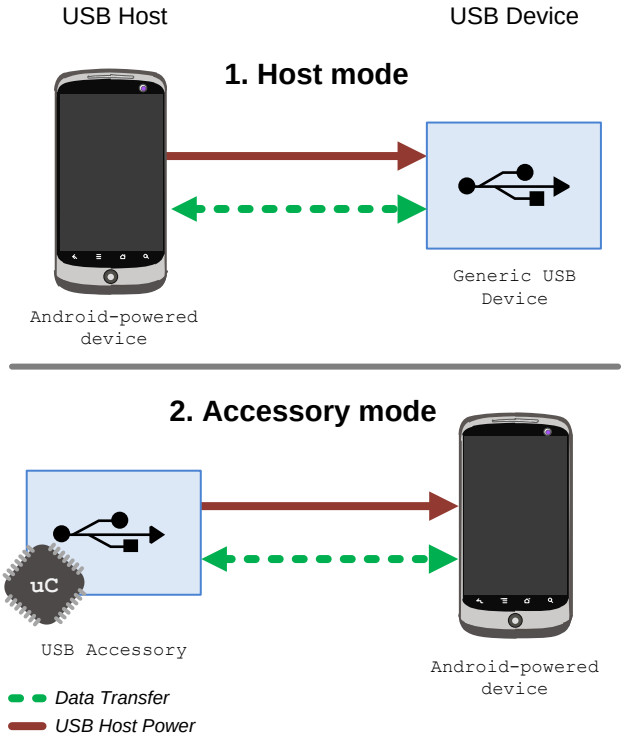


FIGURE 4.2 – Modes USB Host et USB Accessory

4.2.1 Compatibilité

Le tableau ci-dessous résume la prise en charge des modes et protocoles USB supportés par les téléphones et tablettes Android :

TYPE	ANDROID	PARTS DE MARCHÉ	**ADK**	ADB	USB HOST
Téléphones	1.5 à 2.3.3	30%		✓	
	2.3.4 à 2.3.7	65%	✓	✓	
	4 et plus	3%	✓	✓	✓
Tablettes	3.1 et plus	3%	✓	✓	✓

TABLE 4.1 – Compatibilité des modes USB sur Android

Comme déjà mentionné, le protocole ADB est compatible avec tous les téléphones Android sans exception. Officiellement, il n'est malheureusement pas accessible aux développeurs.

Seuls les « anciens » téléphones qui disposent des versions 2.3.3 et inférieures ne sont pas compatibles avec le protocole ADK. Ces téléphones sont équipés de versions d'Android publiées il y a plus de deux ans, donc bien avant la spécification du protocole ADK. Ils ne représentent plus que 30% des téléphones du marché. Bien sûr ce pourcentage va diminuer tout au long de ces prochaines années. Tous les nouveaux téléphones sont équipés de versions récentes et la plupart des constructeurs prévoient la mise à jour de leurs téléphones vers Android 4. Les tablettes fonctionnent aujourd'hui avec Android 3 et sont donc entièrement compatibles.

Il est toutefois possible qu'un constructeur ait choisi de ne pas intégrer la libraire Open Accessory, ce qui pourrait rendre un téléphone récent incompatible. Lors du développement de l'application Android, un paramètre permet d'exiger le support de cette librairie. Dans le cas où la librairie n'est pas supportée, l'application ne pourra pas être installée sur le téléphone en question.

4.2.2 Choix du mode USB

Le premier mode « USB Host » propose d'utiliser le téléphone comme Host. Ce mode n'est supporté que par les tablettes et les téléphones récents équipés d'Android 4, soit 6% des appareils Android ce qui est bien trop peu. Pour communiquer avec Android, le device USB doit implémenter une « Class USB [2] » spécifique, correspondant à son utilisation. L'avantage de ce système est qu'il est très générique. Il peut donc être utilisé avec Android, mais aussi avec tous les PC qui disposent des drivers USB nécessaires, bien que ce but ne soit pas recherché. La classe *Audio Device* devra par exemple être implémentée pour développer une carte son USB. La documentation des différentes classes USB est très dense. Ces classes sont génériques et par conséquent complexes à implémenter. Pour ces différentes raisons, le mode USB Host ne sera pas retenu.

Avec le nouveau mode Accessory, les appareils Android sont utilisés comme Device et c'est l'accessoire, le master (Host) du bus (voir figure 4.2). Cette solution permet donc de rendre compatible plus de 70% des téléphones et tablettes du marché. Le protocole offre aussi une surcouche qui permet de connaitre facilement le type d'accessoire connecté, s'il est supporté par le téléphone, etc. La communication entre l'application Android et l'accessoire est aussi grandement simplifiée. Seuls les téléphones disposant d'Android 2.3.4 et plus sont compatibles avec le protocole Android Accessory. Malgré cette contrainte, c'est ce protocole qui va être utilisé car il prend en charge le plus grand nombre d'appareils. Il a été spécialement conçu pour communiquer facilement entre un périphérique Android et un accessoire USB. L'implémentation de l'accessoire est donc simplifiée et toute la documentation est disponible gratuitement et sans restriction sur le site officiel d'Android. En utilisant ce mode, un stack USB Host Lite sera implémenté sur l'accessoire. C'est un stack limité qui prendra en charge uniquement le protocole ADK. Le support d'autres classes USB ne sera pas implémenté, bien que possible.

4.3 Android Open Accessory version 2

Le protocole Android Open Accessory utilisé est relativement jeune. Comme déjà mentionné, la première version a été présentée en 2011, ainsi que le premier accessoire Android fabriqué par Google. Ce protocole était amené à évoluer. C'est donc logiquement qu'une nouvelle version vient tout juste d'être présentée lors de la même conférence Google I/O qui a eu lieu cette année du 27 au 29 juin 2012.

La version 2 du protocole (AOA 2.0) propose plusieurs nouveautés, telles que le support d'un mode audio. Une nouvelle commande permet d'activer un mode audio sur le périphérique Android. Celui-ci transmet alors automatiquement le flux audio principal du téléphone. Ce mode audio est compatible avec les appareils équipés d'Android 4.1 (*API Level 16*), la dernière version d'Android aussi présentée lors de la conférence 2012. La version 1 de *A0A* reste bien sûr compatible avec les nouveaux appareils Android qui supportent la version 2.

Google a aussi présenté la deuxième version de son accessoire. Toutes les informations concernant Android Open Accessory 2012 et A0A 2.0 sont disponibles sur le site officiel de Android[12]. La mise à jour Android 4.1 est prévue pour le courant du mois de juillet. L'accessoire actuel sera donc compatible uniquement avec la version 1 du protocole.

2. Classes USB officielles - http ://www.usb.org/developers/devclass_docs

Chapitre 5

Android Accessory Board

5.1 Introduction

La carte de développement *uGates* fournie par l'école convient pour les tests et le prototype du produit uniquement. Elle ne répond pas totalement aux besoins du projet. Ce kit n'a pas été prévu pour utiliser le mode USB Host.

La carte *uGates* dispose de plusieurs connecteurs qui permettent d'ajouter facilement des cartes d'extension en « mezzanine ». Cependant, certaines entrées/sorties dont celles de l'USB ne sont pas accessibles. Le connecteur USB de type USB-A ne peut donc être ajouté que difficilement. C'est pourquoi il a été décidé de développer une nouvelle carte électronique plus compacte, conçue spécialement pour l'application d'accessoire USB. Elle intègrera directement les composants nécessaires ainsi qu'un connecteur d'extension pour ajouter d'autres types d'entrées/sorties. Son développement est basé sur la carte *uGates* actuelle et certains composants seront réutilisés.

5.2 Fonctionnalités

Les caractéristiques et fonctionnalités du nouvel accessoire sont les suivantes :

- Alimentation 5V avec un transformateur externe ou alimentation via USB.
- USB Host avec circuit d'alimentation dédié et protection contre les courts-circuits.
- OnBoard JTAG [1] et UART via USB basé sur le *FT2232D*.
- Sortie audio stéréo avec amplificateur pour sortie casque/ligne.
- RTC avec batterie.
- Boutons, LEDs.
- Connecteur d'extension pour composants additionnels.

L'accessoire pourra être alimenté avec une alimentation 5V externe ou via un connecteur USB-B mini. L'appareil Android pourra être connecté via un port USB-A disposé à l'avant. Un JTAG a été intégré directement sur la carte. Elle est programmable via le port mini USB-B. Une sortie UART est aussi disponible via ce port. Une liaison RS-232 n'est donc plus nécessaire.

L'accessoire dispose aussi d'une sortie audio amplifiée. Elle est commandée par un convertisseur DAC audio I2S qui permettra d'écouter de la musique par exemple, via Android. Un *Real-Time Clock* permet de synchroniser l'horloge du téléphone avec celle de l'accessoire pour un réveil ou une alarme par exemple. L'horloge de l'accessoire est maintenue à jour grâce à une pile bouton dédiée.

Des capteurs spécifiques ou un écran peuvent être ajoutés grâce à un connecteur d'extension. Comme la carte *uGates*, le nouvel accessoire possède différents boutons, LEDs et une LED RGB.

1. Joint Test Action Group - http ://en.wikipedia.org/wiki/Joint_Test_Action_Group

5.3 Choix du processeur

Les développements ont été testés sur le processeur *LPC1768* de la carte *uGates*, un Cortex M3 de NXP. L'idée est de garder la même famille de processeurs NXP, car elle convient parfaitement pour notre application. Ainsi, les développements sont réutilisables sans modification.

Le processeur LPC1768 dispose de plusieurs interfaces qui ne sont pas utilisées, notamment l'interface Ethernet. En effet, cette interface n'est pas nécessaire pour la docking station puisque le téléphone Android dispose déjà d'une interface réseau (3G, ou WiFi). Ce processeur est disponible en TQFP100. Il dispose donc de plus de 70 GPIO, dont la majeure partie n'est pas utilisée pour cette application.

La famille des LPC17 est composée de 31 processeurs. Le tableau ci-dessous regroupe les processeurs qui disposent d'un contrôleur USB Host et qui disposent de moins de 100 pins. Le prix est indicatif pour une pièce :

Type	Fmax	Flash	RAM	Ethernet	USB Host	Package	Prix
LPC1754	100MHz	128kB	32kB	0	1	LQFP80	CHF 8.-
LPC1756		256kB		0			CHF 10.-
LPC1758		512kB	64kB	1			CHF 13.-
LPC1759	120MHz			0			CHF 20.-
LPC1765	100MHz	256kB	64kB	0	1	LQFP100	CHF 10.-
LPC1766				1			CHF 10.-
LPC1768		512kB		1			CHF 13.-
LPC1769	120MHz			1			CHF 15.-

Table 5.1 – Comparaison des processeurs pour l'accessoire USB

Les processeurs LPC175X disposent de 52 E/S, ce qui est suffisant pour l'accessoire et qui permet de réduire la taille de la carte électronique et de simplifier sa fabrication. Le choix s'est porté sur le processeur *LPC1758* qui dispose de suffisamment de mémoire flash et de ram pour développer des applications C++ sans limitation (32kB de ram peut s'avérer insuffisant). La version avec 64kB de ram a été choisie pour les tests. Le LPC1758 dispose d'un contrôleur Ethernet interne qui ne sera pas utilisé. Le LPC1759 n'a pas été choisi car son prix est trop élevé. Sa fréquence de travail élevée n'est pas nécessaire.

Les processeurs LPC175X sont pins compatibles. Pour la version finale, il est envisageable de passer au processeur LPC1756 ou 54 afin de réduire les coûts.

5.4 Fabrication

Le PCB de l'accessoire a été développé sur P-CAD 2006. Différents composants ont été créés. Le PCB reste un prototype. Des points de mesure ont donc été prévus. Les différentes fonctions (blocs) du PCB peuvent être testées individuellement, voire bipassées à l'aide de ponts.

Le routage a été réalisé en interne par l'atelier PCB. La fabrication de la carte en interne est complexe du fait de la taille réduite des composants utilisés. La carte électronique ne sera pas fabriquée en interne, bien que cette solution ait été envisagée. EuroCircuits propose la fabrication de PCB professionnels quatre couches. Le routage de la carte a donc été grandement simplifié. La taille de l'accessoire n'est pas importante (prototype), mais elle a pu être considérablement réduite en choisissant une fabrication professionnelle. Plusieurs jours de travail ont pu être gagnés en choisissant cette solution.

Trois PCB ont été commandés pour un total de 100 € environ. La taille du produit final (110x90mm) pourra encore être réduite :

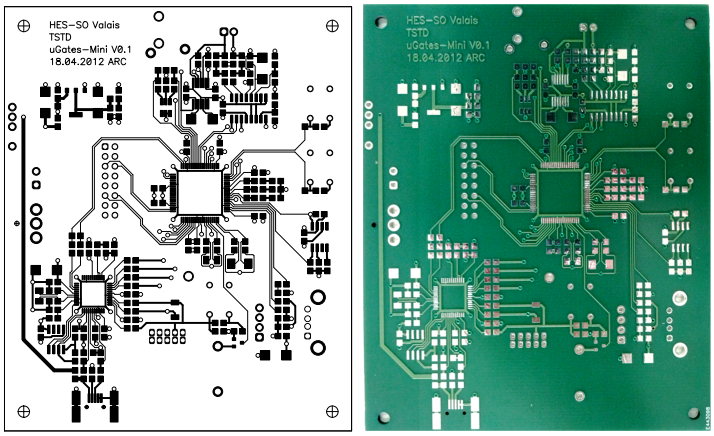


FIGURE 5.1 – PCB du nouvel accessoire USB (top layer, pas à l'échelle)

Le dossier de fabrication (schéma, PCB, commande de composants, etc.) du PCB est disponible en annexe E.

5.5 Accessoire *uGates-mini*

L'accessoire USB *uGates-mini* est présenté dans la figure suivante :

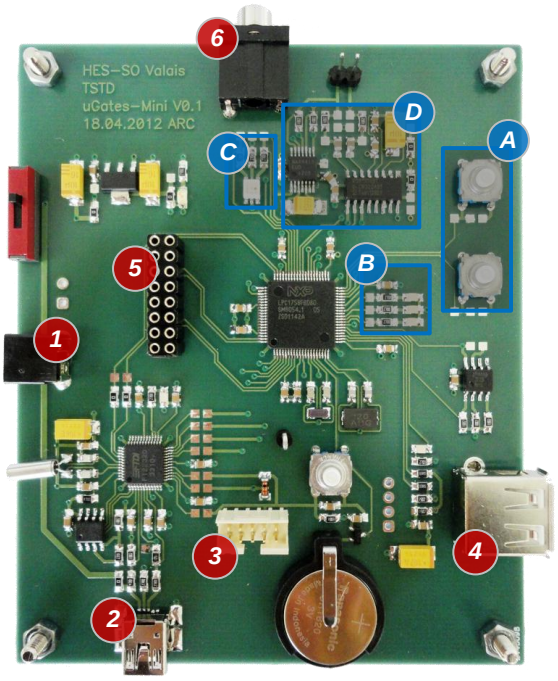


FIGURE 5.2 – Android Accessory Board

Plusieurs indications y figurent. Tout d'abord, les nombres en rouge concernent les connecteurs de la carte :

1 Connecteur jack d'alimentation pour transformateur externe 5 VDC.

2 Connecteur mini-USB pour JTAG intégré et sortie UART. La carte peut aussi être alimentée par USB. Lorsqu'un périphérique Android est connecté, le courant maximum peut dépasser les 500mA disponibles par USB. L'alimentation externe doit alors être utilisée.

3 Connecteur pour JTAG externe.

4 Connecteur USB-A pour la connexion des périphériques Android.

5 Connecteur d'extension 16 pôles pour des cartes mezzanines additionnelles.

6 Connecteur audio stéréo (jack 3.5mm).

Les lettres de couleur bleue de A à D concernent les ressources disponibles directement sur la carte, soit respectivement deux boutons, trois LEDs jaunes, une LED RGB et la sortie audio stéréo amplifiée. Des ressources additionnelles peuvent être ajoutées via le connecteur d'extension (voir chapitre 5.10 page 29).

Pour rappel, tout accessoire USB qui utilise le protocole Android Open Accessory doit pouvoir être identifié par les périphériques Android. La carte *uGates-mini* possède les paramètres d'identification suivants :

Accessoire USB uGates-mini	
Fabricant	HES-SO Valais
Modèle	uGates-mini
Description	uGates-mini Accessory
Version	1.1
URI	http ://www.hevs.ch
Numéro de série	0123456789

Table 5.2 – Identification de l'accessoire *uGates-mini*

Liste des entrées/sorties

Le tableau ci-dessous résume les différentes E/S disponibles sur l'accessoire *uGates-mini* :

ID	Nom	Type	Pin	Description
0	led0	*DigitalOutput*	P1#23	LED verte 0
1	led1		P1#24	LED verte 1
2	led2		P1#26	LED verte 2
3	switch0	DigitalInput	P0#10	Bouton 0
4	switch1		P0#11	Bouton 1
5	ledRGB0	*LedRGB*	-	LED RGB
6	uart0	*Uart*	-	Module Uart
7	rtc0	*RTC*	-	Real Time Clock
8	pot0	*AnalogInput*	P0#2	Potentiomètre
9	lcd0	*Lcd*	-	Ecran LCD graphique
128	audio0	Audio	-	Sortie audio stéréo

Table 5.3 – Ressources disponibles sur l'accessoire

Ces E/S peuvent être commandées depuis le périphérique Android connecté à l'accessoire. Elles sont vues comme des « ressources », qui possèdent un identifiant unique et un nom. La notion de ressource est expliquée plus en détail dans le chapitre 6.

Des E/S basiques tels que LEDs, boutons sont disponibles sur le kit. Il dispose aussi d'un RTC qui, une fois synchronisé avec l'horloge d'un téléphone Android, pourrait par exemple servir de réveil ou d'alarme. L'écran LCD graphique permet d'afficher l'heure, le modèle du téléphone connecté, etc. Enfin, la sortie audio ajoutée sur la carte *uGates-mini* va permettre à l'utilisateur de diffuser sa propre musique sur des haut-parleurs.

Chaque accessoire dispose d'un hardware spécifique. Les E/S disponibles devront donc être décrites dans un fichier de configuration qui sera utilisé par le téléphone afin de développer des applications génériques (voir chapitre 7.5) sur Android.

5.6 CMSIS

Les développements de l'accessoire *uGates-mini* ont été réalisés en C/C++. La couche d'abstraction *Cortex Microcontroller Software Interface Standard[2]* (CMSIS) facilite l'accès au matériel. Les contrôleurs hardware suivants ont été utilisés :

- GPIO
- Convertisseur A/D
- Contrôleur SPI pour l'écran LCD
- Contrôleur PWM pour la led RGB
- RTC
- Uart
- Contrôleur I2S pour la sortie audio
- Contrôleur GPDMA pour les transferts USB - I2S
- Contrôleur USB Host

CMSIS met aussi à disposition tous les fichiers nécessaires à la programmation du processeur, tels que le fichier de startup, le linker script, etc.

5.7 USB Host

Un stack USB Host Lite a été implémenté afin de communiquer via USB et le protocole ADK avec un périphérique Android compatible.

La carte *uGates-mini* dispose d'un microcontrôleur de type Cortex M3 qui intègre un contrôleur USB 2.0 interne. Il peut être utilisé pour l'USB Host, mais aussi pour l'USB Device et OTG. L'USB On-The-Go est une fonctionnalité supplémentaire introduite dans USB 2 afin d'échanger des données point à point entre deux appareils, sans devoir passer par le Host du bus. L'utilisation de cette fonctionnalité nécessite un composant externe (exemple : *ISP1302*) qui fait office d'émetteur/récepteur. Cette norme ne sera pas implémentée. Dans ce projet, un seul device USB pourra être connecté au master. Cette limitation est aussi fixée par le protocole ADK. Le protocole de communication en couche est présenté dans la figure ci-dessous :

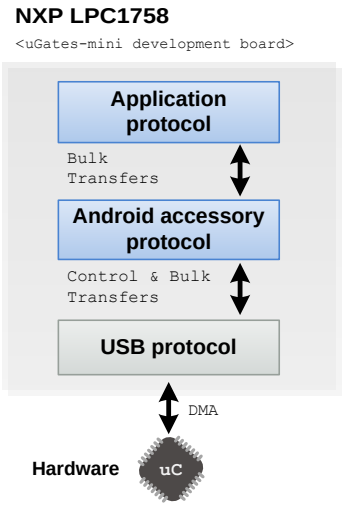


FIGURE 5.3 – Protocole de communication en couche

Le contrôleur USB Host intégré est accessible par la couche physique via DMA. Des trames USB de type *Bulk* et *Control* peuvent être envoyées via cette couche physique. Seuls ces types de trames USB ont été

implémentées. La communication avec Android est établie via des trames de type *Control* spécifiées par le protocole Android Accessory. Une fois la communication établie, des trames *Bulk* sont envoyées entre l'accessoire et Android pour échanger des données. Ces différentes couches sont détaillées séparément dans les chapitres suivants.

5.7.1 Câblage USB Host

Un seul contrôleur USB est disponible avec les microcontrôleurs *LPC17XX*. Il peut fonctionner selon différents modes USB (Host, Device, OTG). Un seul d'entre-eux peut être utilisé simultanément. De plus, les schémas de câblage du contrôleur dépendent du mode utilisé. Pour fonctionner en mode USB Host, le contrôleur doit être connecté de la manière suivante :

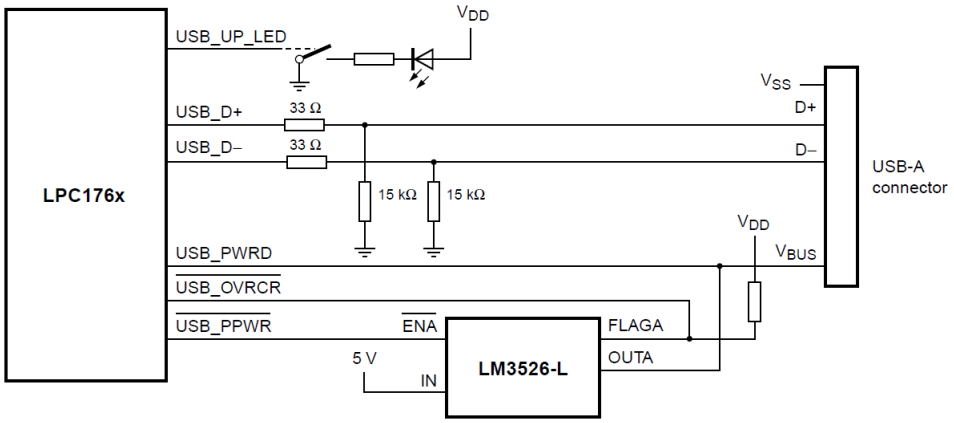


FIGURE 5.4 – Câblage du contrôleur USB Host

Le connecteur USB de type A permet de connecter le périphérique via un câble USB standard. Le master USB doit fixer le niveau logique du bus de données différentiel à l'aide des deux résistances de 15k Ohm. Ces résistances sont indispensables et permettent de détecter correctement la déconnexion du périphérique USB.

Les fonctions des E/S du contrôleur USB sont résumées ci-dessous :

NOM	DIRECTION	DESCRIPTION	TYPE
USB_D+	bidirectionnel	Donnée différentielle +	Connecteur USB
USB_D-	bidirectionnel	Donnée différentielle -	Connecteur USB
USB_UP_LED	sortie	Pin de contrôle	Contrôle
USB_PPWR	sortie	Activation du port USB	Alimentation
USB_PWRD	entrée	Statut du port USB	Alimentation
USB_OVRCR	entrée	Indique une surcharge de courant	Alimentation

TABLE 5.4 – Pins du contrôleur USB Host

Le host USB doit aussi fournir l'alimentation 5V pour le périphérique USB. Si le périphérique n'est pas auto-alimenté, il doit pouvoir être rechargé et peut nécessiter jusqu'à 500mA. L'utilisation d'un contrôleur d'alimentation externe est recommandée. Connecté aux signaux d'alimentation présentés ci-dessus, le LM3526 (voir figure 5.4) a été spécialement conçu pour ce contrôleur et permet de couper l'alimentation du port USB en cas de court-circuit ou de surcharge de courant.

Le contrôleur dispose d'une sortie de contrôle qui peut être câblée sur une LED. Elle est gérée directement par le contrôleur et indique l'état de la connexion USB.

5.7.2 Stack USB Host Lite

Le contrôleur USB est disponible dans différents processeurs de la famille LPC17 : LPC1768, LPC1766, LPC1765, LPC1758, LPC1756 et LPC1754. Son contrôleur USB est conforme à la norme *OHCI*[17] et est accessible via un accès direct à la mémoire (*DMA*). Elle spécifie les registres disponibles ainsi que les différentes fonctions implémentées par le contrôleur. Les fonctions USB développées sont basées sur cette documentation.

Un projet de base[4] intégrant un contrôleur USB Host Lite est disponible chez NXP. L'application de démonstration fournie permet d'écrire un fichier texte sur une mémoire flash USB (classe *MassStorage*). Les fonctions « low level » de la stack sont implémentées. Des trames USB de Control et de type Bulk peuvent être transmises. Il est aussi possible de récupérer des descripteurs, définir des configurations, etc. Il a été compilé pour le kit de développement *RDB1768* de Code Red et nécessite la librairie *CMSIS*. Des modifications ont dû être apportées pour pouvoir utiliser le projet avec la carte *uGates* et compiler le projet avec les outils de développement.

Seules les fonctions de la couche physique USB ont été reprises de l'exemple et intégrées dans la classe *USBHost*. Un nouveau projet C++ a en effet été créé. Les fonctions ont été encapsulées dans des classes afin de rendre le code plus lisible et d'offrir un meilleur découplage entre les différentes couches. Plusieurs améliorations ont été apportées à ces fonctions. Fournies à titre d'exemple, elles comportaient plusieurs dysfonctionnements. Par exemple, les méthodes permettant de transmettre des trames USB de type Bulk étaient totalement bloquantes. De plus, il était impératif de connecter un périphérique USB avant la mise en marche du système, sans quoi il n'était pas détecté. Ces défauts ont été corrigés. Des *timeout* ont été ajoutés en cas de non-réponse ou d'erreur de communication avec le périphérique USB.

Un seul périphérique USB peut être connecté simultanément. Une machine d'états pour le contrôleur USBHost a été implémentée. Elle est dessinée dans la figure suivante :

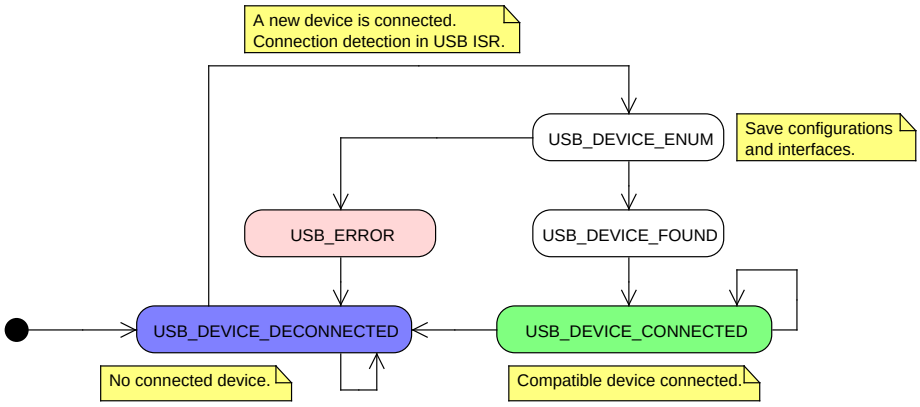


FIGURE 5.5 – Machine d'états du contrôleur USB Host

Par défaut, aucun périphérique USB n'est connecté. Lors de son branchement, une interruption est générée par le contrôleur USB. Cette interruption peut être générée pour plusieurs raisons. Il faut donc analyser les registres de statut du contrôleur pour détecter qu'il s'agit bien d'un nouveau périphérique connecté.

L'état *USB_DEVICE_ENUM* énumère les configurations et les interfaces du périphérique. La méthode *parseConfiguration* parcoure les configurations du périphérique (voir figure 3.3) et permet de sauver deux Endpoint USB d'entrée/sortie de type Bulk. Ces deux Bulk Endpoint de l'interface 1 sont sauvés et permettront par la suite de communiquer avec le périphérique via le protocole Android Accessory. La configuration 1 du périphérique doit être utilisée et peut être sélectionnée via une requête USB de type *SET_CONFIGURATION (0x09)*. Il est important d'utiliser les deux Endpoint Bulk de la première interface pour communiquer avec le téléphone via ADK. D'autres Endpoint sont disponibles pour le protocole ADB, la classe USB MassStrorage, etc.

Si les deux Endpoint ont pu être trouvés et sauvés, l'état *USB_DEVICE_FOUND* affiche les informations principales du périphérique connecté, comme l'ID du fabricant, du vendeur, le type et le numéro de série du périphérique, etc. Dès à présent, le périphérique USB est considéré comme compatible et connecté.

Si la configuration du périphérique USB n'est pas compatible, une erreur peut être affichée via l'état *USB_ERROR*.

Lorsque l'appareil USB est déconnecté, la machine d'états passe dans le mode *USB_DEVICE_ DECONNECTED* et le processus décrit ci-dessus peut redémarrer.

5.7.3 Communication avec Android

L'accessoire et le périphérique Android communiquent par USB via le protocole Android Accessory. Cette couche de communication utilise des trames USB de type Control et Bulk ainsi que les Endpoint sauvés par la couche USB.

La documentation Android décrit les différentes étapes nécessaires à l'activation du mode Accessory sur le téléphone et avant tout détecter si ce dernier est compatible. Les étapes sont résumées ci-dessous :

1. Détection d'un périphérique USB.
2. Déterminer si le mode Accessory est supporté par l'appareil.
3. Démarrage du mode Accessory (si nécessaire).
4. Etablir la communication avec l'appareil.

Avant que la communication soit établie, seules des commandes USB de types Control peuvent être envoyées au périphérique. La tâche numéro 1 est réalisée par la machine d'états de la couche USB (voir figure 5.5). Un périphérique USB est connecté lorsqu'elle est dans l'état *USB_DEVICE_CONNECTED*.

Si c'est le cas, il faut détecter si le mode Accessory est supporté par le périphérique. Une commande du protocole Accessory (commande USB de type Control) permet de connaitre la version du protocole compatible. A ce jour, seule la version 1 est supportée. Les informations identifiant l'accessoire doivent ensuite être envoyées. Une commande ADK est prévue à cet effet. Les informations envoyées sont des chaines de caractères qui doivent se terminer par '\0' et ne pas dépasser 256 caractères. Un index les identifie selon le tableau suivant :

Informations	Index	Exemple
Fabricant	0	HES-SO Valais
Modèle	1	uGates-mini
Description	2	uGates-mini Accessory
Version	3	1.1
URI	4	http ://www.hevs.ch
Numéro de série	5	0123456789

Table 5.5 – Informations d'identification d'un accessoire USB

Une fois ces informations envoyées, il est possible de démarrer le mode Accessory du périphérique Android. L'utilisateur verra alors apparaître un dialogue à l'écran pour valider le démarrage du mode (plus de détails dans le chapitre suivant). Le périphérique va se déconnecter puis se reconnecter et sera vu cette fois-ci comme un appareil « Google ». Sa nouvelle configuration USB est réexaminée par la machine d'états USB. Un *vendorID* et un *productID* spécifiques permettent d'identifier que le périphérique est bien en mode Accessory.

De manière visuelle, voici comment le mode Accessory peut être activé. Les étapes sont numérotées dans le diagramme suivant :

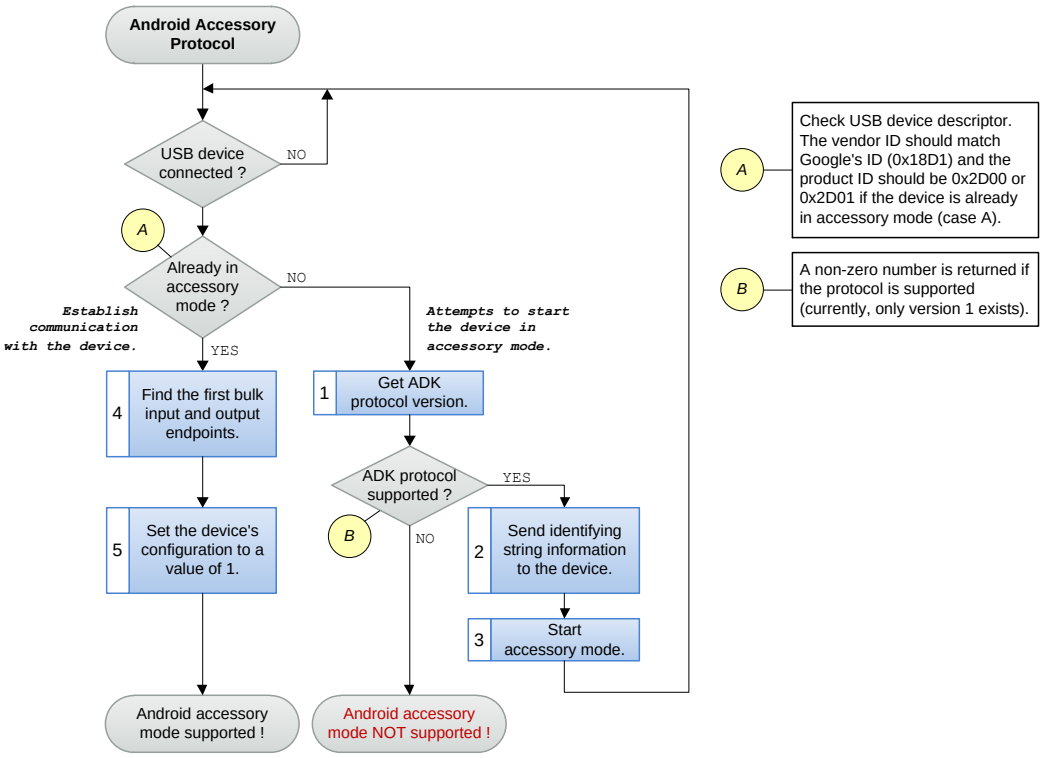


FIGURE 5.6 – Support du protocole Android Accessory

Lorsque le mode Accessory est activé sur le téléphone, des données peuvent être envoyées/reçues via des commandes de type Bulk. Lorsqu'on le déconnecte, le mode Accessory est automatiquement stoppé. Ce diagramme a été implémenté à l'aide de la machine d'états suivante :

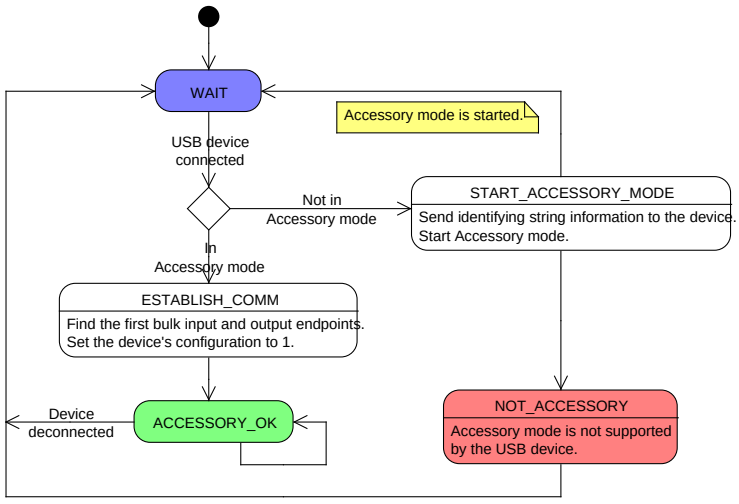


FIGURE 5.7 – Machine d'états pour le mode Accessory

Le détail des commandes USB de type Control utilisées précédemment, tels que les types, index et valeurs sont disponibles dans la documentation officielle d'Android[12].

5.8 JTAG intégré

Le JTAG intégré est basé sur le *FT2232D*, un dual USB UART/FIFO IC utilisé dans beaucoup de sondes JTAG, dont le JTAGkey [2] qui est utilisé comme sonde externe dans le projet.

Le *FT2232D* dispose de deux canaux (ports A et B), configurables indépendamment. Le canal A est configuré en mode FIFO pour le JTAG. Le canal B est connecté au périphérique UART du processeur et est configuré en mode RS232 UART.

Les configurations du FTDI sont les suivantes :

Paramètre	Valeur
Port A	CPU FIFO
Port B	RS232 UART
Fabricant	HES-SO Valais
Description	Android Accessory Board
Numéro de série	Généré automatiquement

Table 5.6 – Configuration du FTDI

Les configurations du FTDI peuvent être modifiées via le logiciel FT Prog [3] fourni par le fabricant. Le *vendorID* du chip n'a pas été modifié et il est préférable de ne pas le modifier. Ainsi, le driver générique FTDI est automatiquement téléchargé lors de son branchement sur PC.

A ce jour, seule la sortie UART a été testée. Elle a été utilisée principalement pour du debug. Une sonde JTAG externe a été utilisée pour la programmation du processeur, ce qui a permis d'accélérer les tests de la carte. Le JTAG intégré sera testé durant l'été. A noter que la sonde externe ne peut pas être utilisée en même temps que le JTAG intégré.

5.9 Sortie audio

Une sortie audio a été ajoutée sur l'accessoire *uGates-mini*. Elle permet à l'utilisateur d'écouter de la musique via le périphérique Android. Voici le schéma bloc de cette sortie audio stéréo amplifiée :

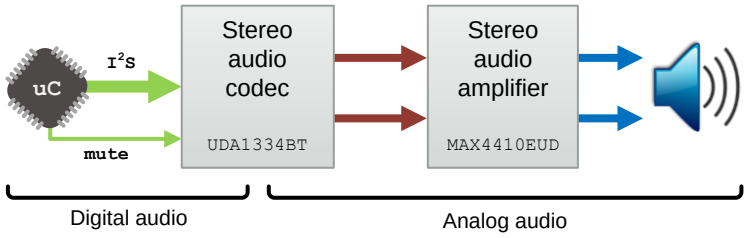


Figure 5.8 – Schéma bloc de la sortie audio stéréo amplifiée

Les samples audio PCM sont tout d'abord envoyés au codec (*UDA1334BT*) qui les convertit en audio analogique stéréo, en rouge sur le schéma ci-dessus (canaux droit et gauche). Le codec audio dispose d'une entrée mute qui permet de désactiver la sortie audio. Il supporte des fréquences audio de 8 à 100kHz. Pour cette première version du projet, la fréquence de la sortie audio a été limitée à 16kHz, pour faciliter la programmation et les tests de la sortie audio.

Elle est ensuite amplifiée via le *MAX4410EUD* (80mW). Cette amplification permet de connecter directement des écouteurs ou des haut-parleurs via le jack 3.5mm. La sortie audio peut aussi être déportée. L'accessoire ne dispose pas de haut-parleurs intégrés.

2. Amontec JTAGkey - http ://www.amontec.com/jtagkey.shtml
3. EEPROM Programming Utility - http ://www.ftdichip.com/Support/Utilities.htm

5.10 Carte d'extension

Une carte additionnelle a été développée afin d'ajouter des entrées/sorties à l'accessoire de base. Elle utilise le connecteur d'extension 16 pôles prévu à cet effet. Développée à des fins de démonstrations, elle intègre les éléments suivants :

- Un écran LCD graphique noir/blanc avec rétro-éclairage blanc.
- Un potentiomètre linéaire utilisé comme entrée analogique.
- Deux LEDs.

La carte « *MezzaLCD* » peut être déportée de l'accessoire pour l'ajout d'un éventuel boitier mécanique. Les deux LEDs de la carte ont été initialement prévues pour indiquer l'alimentation de la carte ainsi que l'état de connexion du périphérique Android. Elles ne sont donc pas directement accessibles via Android. Le rétro-éclairage de l'écran peut aussi être commandé par le processeur. Connecté à une sortie PWM, sa luminosité peut varier.

La carte d'extension a été fabriquée à l'école. Elle se présente sous cette forme :

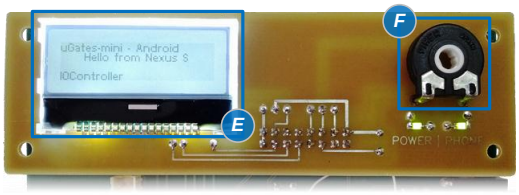


Figure 5.9 – Carte d'extension

L'écran LCD (lettre E) et le potentiomètre (lettre F) sont les ressources de la carte d'extension qui sont accessibles via Android.

La carte est alimentée en 3.3V. Les entrées/sorties utilisées sont résumées dans le tableau ci-dessous :

Pin	Direction	Nom	Description
CONN13 - P1#20	Sortie	*LCD_LIGHT*	Rétro-éclairage écran *PWM1.2*
CONN12 - P0#26	Sortie	*LCD_A0*	Data LCD
CONN14 - P2#02	Sortie	*#LCD_RST*	Reset LCD
CONN06 - P0#03	Sortie	*#LCD_CS*	Chip select LCD
CONN03 - P0#00	Sortie	*LED0*	LED *Phone*
CONN05 - P0#02	Entrée	*AD07*	Entrée analogique *AD0.7*
CONN07 - P0#15	Sortie	*SCK0*	Clock SPI
CONN10 - P0#18	Sortie	*MOSI0*	Data SPI

Table 5.7 – E/S de la carte mezzanine

Le dossier complet de fabrication de la carte est disponible en annexe F.

Écran LCD

La carte mezzanine dispose d'un écran LCD qui sert d'interface utilisateur. Lorsqu'aucun appareil n'est connecté, l'heure qui provient du RTC est affichée. Lorsqu'une application contrôle l'accessoire, l'utilisateur a pleinement accès à l'écran pour y afficher différentes informations.

L'écran choisi est un écran LCD graphique 128x32 pixels. Il est commandé par SPI et dispose d'un rétro-éclairage blanc intégré. Il a été choisi pour son bas coût (environ 10 CHF). Sa taille plutôt faible permet tout de même d'afficher toutes les informations utiles, comme le montre la figure ci-dessous :

FIGURE 5.10 – Écran LCD graphique avec rétro-éclairage

Avec cet écran graphique, il est aussi possible de dessiner toutes sortes de symboles et de formes. Pour ce travail, un contrôleur d'écran simple a été implémenté. L'écran ne disposant pas de table de caractères intégrée, une table a donc été générée en utilisant le logiciel open-source TheDotFactory [4]. Tous les caractères et chiffres de la police Arial 7pt ont été générés. Pour simplifier les méthodes de dessin, l'écran a ensuite été divisé en 4 lignes (pages) de 8 pixels. L'utilisateur peut ainsi écrire des textes en spécifiant le numéro de ligne (0 à 3) ainsi que la coordonnée x (0 à 128). Le contrôleur LCD reste basique. Si besoin est, des fonctions de dessin plus complexes pourront être implémentées.

5.11 Modifications

Pour que l'accessoire *uGates-mini* fonctionne correctement dans sa version actuelle, trois modifications ont dû être apportées :

1. Connexion de la pin n°62 *I2STX_WS* du processeur *LPC1758* à la pin n°2 *WS* du codec audio *UDA1334*.
2. Ajout d'un condensateur de 2.2µF entre les pins n°6 *PVSS*, n°7 *SVSS* de l'amplificateur audio *MAX4410EUD* et le *GND* selon le schéma ci-dessous :

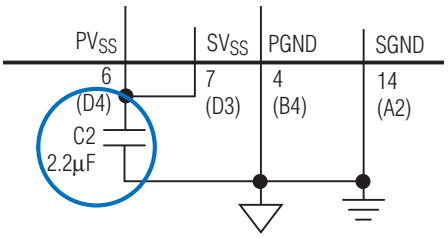


FIGURE 5.11 – Correction PCB pour amplificateur audio

3. Ajout d'une horloge externe sur la pin n°6 du codec audio *UDA1334BT*.

Le codec audio nécessite un clock externe, en plus du clock I2S. Les informations disponibles dans le datasheet étaient peu claires à ce sujet. La fréquence du clock externe dépend de la fréquence du clock I2S. Elle doit être un multiple exact de 128, 192, 256, 348, 512 ou 768 fois le clock I2S.

Actuellement, la sortie audio est configurée à 16kHz. Le clock I2S a été mesuré et est de 16.275kHz. Le calcul ci-dessous permet de déterminer la fréquence d'un clock compatible :

4. An LCD Font and Image Generator - http ://www.pavius.net

Algorithme 5.1 Calcul du clock externe pour le codec audio.

$$SYSCLK = f_{WS} \cdot 128 = 16,275kHz \cdot 256 = 4.1664MHz$$

Un générateur de fonction est utilisé pour générer précisément cette fréquence. Si la qualité audio change, le clock externe doit bien sûr être adapté.

Une autre version du codec audio est disponible, avec cette fois-ci une PLL intégrée. La PLL permet de généré la fréquence calculée ci-dessus à l'aide du clock I2S. La version *UDA1334ATS* n'est pas disponible dans le même package que le codec audio actuel. Le changement du codec audio devra être réalisé dans la prochaine version de l'accessoire :

– Changement du codec audio *UDA1334BT* pour la version *UDA1334ATS* avec PLL intégrée.

Pour la démonstration, un générateur de fonction externe est utilisé. Si la fréquence générée n'est pas assez précise, on peut entendre des légers « pop ».

Dans la nouvelle version de l'accessoire, le générateur ne sera plus nécessaire. La qualité audio sera augmentée, car le clock externe sera en phase et synchronisé avec le clock I2S. De plus, le codec audio pourra s'adapter à la qualité audio choisie.

L'image ci-dessous présente les modifications apportées :

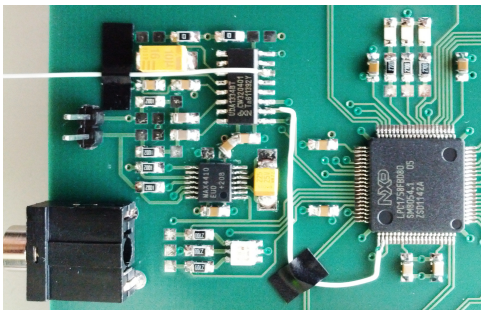


FIGURE 5.12 – Modifications apportées à l'accessoire *uGates-mini*

Aucun test n'avait pu être effectué avant la fabrication de cette première version. A ce stade, l'accessoire est entièrement fonctionnel et les quelques modifications seront apportées dans la prochaine version.

Chapitre 6

Couche d'abstraction matérielle

Le protocole Android Open Accessory déjà décrit permet de transférer facilement des données entre l'accessoire et le périphérique Android via des trames USB *Bulk*. Ce chapitre traite du protocole de communication plus « haut niveau » utilisé entre ces deux périphériques, comme le montre une nouvelle fois la figure ci-dessous :

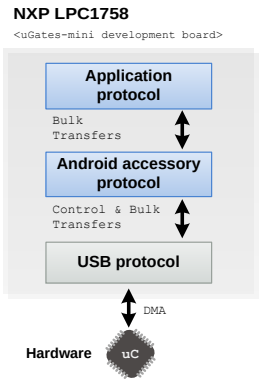


FIGURE 6.1 – Protocole de communication en couche

La couche « *Application protocol* » est utilisée pour construire une vue haut niveau des données échangées et sert d'abstraction matérielle. Elle est générique et ne dépend ni du type d'accessoire USB ni du type de périphérique Android connecté.

6.1 Introduction

Les données échangées entre l'accessoire et le périphérique Android sont assignées à des ressources (*Resource*). Elles identifient donc les types de données qui sont partagées. Cinq types ou interfaces ont été définis, comme le montre la figure ci-dessous :

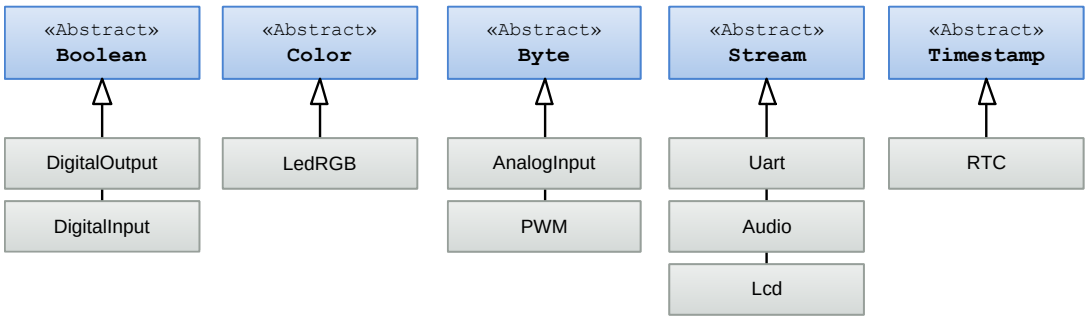


FIGURE 6.2 – Types de ressources à disposition

Les types de ressources sont définis plus précisément ci-dessous :

Type	Taille	Description
Boolean	1 byte	Valeur binaire (*true/false*) pour indiquer l'état d'un bouton ou d'une LED.
Color	3 bytes	Composantes RGB d'une couleur (1 byte par couleur).
Byte	1 bytes	Valeur entière codée sur 1 byte.
Stream	n bytes	Données de longueur variable (textes, sons, etc.).
Timestamp	7 bytes	Date et heure codée en String (exemple : *01.05.1990 - 18h30 :22s*).

Table 6.1 – Description des types de ressources

Ces types de données peuvent être comparés à des structures de données de taille prédéfinies, en fonction du type de l'information à transmettre. Par exemple, une valeur entière 8 bits (de type *Byte*) peut servir à transmettre la valeur d'une entrée analogique ou le duty cycle d'une sortie PWM. De ce fait, l'utilisateur ne pourra pas directement créer une ressource de type *Byte*, *Boolean*, etc. Ces ressources sont des classes abstraites qui ne peuvent pas être instanciées mais qui servent de base pour leurs sous-classes. La deuxième utilité de ces classes abstraites est de fournir une interface « type safe » pour les classes qui en héritent.

Le diagramme de classes complet qui présente les ressources et les types disponibles se trouve en annexe G du rapport.

6.2 Ressources

Les ressources, tels que LEDs, boutons, écrans et autre E/S de l'accessoire USB sont identifiées par un numéro (ID) unique ainsi qu'un type. Les types de ressources sont prédéfinis dans une énumération (*ResourceType*) et dépendent des différentes interfaces disponibles.

Chaque ressource dispose de deux méthodes qui leur permet d'envoyer (accessoire vers Android) et de recevoir des données via la liaison USB :

```
1 // Pure virtual method. Must be implemented in subclass.
2 virtual void doJob(uint8_t *data) = 0;
3
4 // Return null if the resource has nothing to send over USB.
5 virtual USBData newData(void) = 0;
```

Listing 6.1 – Déclaration d'une ressource

Lorsque des données USB sont réceptionnées sur l'accessoire, la méthode *doJob* de la ressource concernée est appelée. La ressource concernée est connue grâce a son identifiant, toujours transmis au début de la commande, comme le montre la figure suivante :

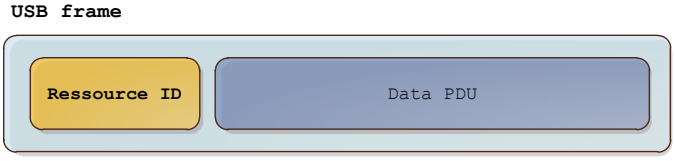


Figure 6.3 – Format des trames USB

La fonction *doJob* transmet ensuite le pointeur vers les données (Data PDU) et c'est ensuite à la ressource elle-même de les décoder. La longueur et le type de données sont connus grâce au type de la ressource. La méthode *doJob* doit donc être implémentée pour les cinq types de ressources. Ce scénario de réception de données est décrit plus en détail dans la section suivante.

La fonction *newData* permet cette fois-ci aux ressources d'envoyer des données sur l'USB. La fonction *newData* de chaque ressource est appelée périodiquement afin de savoir si l'une d'entre-elles a des nouvelles données à transmettre. Si c'est le cas, la longueur et les données sont retournées pour être envoyées. Par défaut, lorsqu'un changement d'état est détecté sur une entrée, sa nouvelle valeur est automatiquement transmise vers le périphérique Android. Le code de la classe *Ressource* est disponible en annexe H.

6.3 Types de ressources

Le tableau ci-dessous regroupe les types de ressources de l'accessoire *uGates-mini*. Les actions de lecture et d'écriture sont résumées pour les différents types :

Nom	Type	Fonction doJob	Fonction newData
DigitalOutput	*Boolean*	Commande booléenne de la sortie.	Lecture de la dernière valeur de la sortie (copie en cache).
DigitalInput	*Boolean*	-	Lecture booléenne de l'entrée.
Uart	*Stream*	Envoi du message par UART.	Lecture de caractères non-implémentée.
PWM	*Byte*	Commande du duty cycle de la sortie PWM.	Lecture de la dernière valeur de la sortie (copie en cache).
LedRGB	*Color*	Modification des duty cycle des trois sorties PWM pour modifier la couleur de la LED.	Lecture de la dernière valeur de la sortie (copie en cache).
RTC	*Timestamp*	Mise à jour du timestamp.	Retourne le timestamp actuel.
Lcd	*Stream*	Affichage du texte à la position souhaitée.	-
Audio	*Stream*	Envoi des samples audio vers le codec externe.	-

Table 6.2 – Types de ressources de l'accessoire *uGates-mini*

Certains types de ressources sont des entrées, d'autres des sorties. Il est néanmoins possible de lire la valeur d'une sortie. C'est en fait la dernière valeur de celle-ci qui est retournée. Ces cas particuliers sont représentés en gris dans le tableau.

La liste complète des E/S sorties disponibles sur l'accessoire *uGates-mini* se trouve au chapitre 5.5.

Une commande USB a été définie pour chaque type de ressources. Le détail de ces commandes est disponible en annexe I.

6.4 Réception des données sur l'accessoire USB

Le scénario ci-dessous présente le traitement des données USB reçues sur l'accessoire. La trame de 4 bytes ci-dessous a été choisie comme exemple. Elle permet de commander la couleur de la LED RGB, en l'occurrence la couleur bleue. La LED RGB est identifiée par l'ID unique 5 (voir chapitre 5.5). Le scénario s'applique pour toutes les ressources, seul le format des commandes varie.

Figure 6.4 – Commande de la LED RGB

On admet que le périphérique Android est compatible et correctement connecté. La trame ci-dessus est envoyée à l'accessoire. Voici comment elle est traitée par ce dernier :

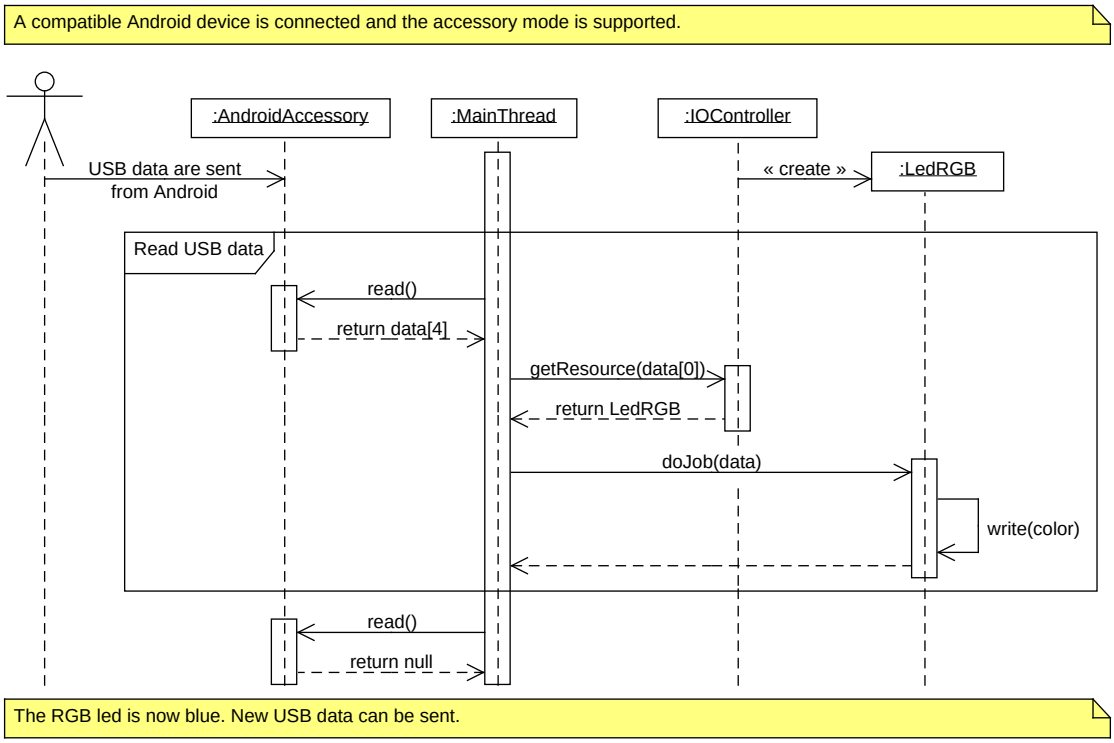


FIGURE 6.5 – Scénario de réception de données sur l'accessoire USB

On retrouve tout à gauche le périphérique Android, vu comme acteur externe au système. Le *Thread* principal *MainThread* est en fait une boucle sans fin, puisque le système ne dispose pas de système d'exploitation. La classe *IOController* gère les ressources de l'accessoire USB. Elles y sont toutes instanciées de manière statique et sont accessibles via leur ID unique.

L'utilisateur sélectionne la couleur bleue pour la LED RGB, les données sont envoyées par le périphérique Android et réceptionnées en RAM par le contrôleur USB de l'accessoire. Par polling, le Thread principal du système va lire les nouvelles données. Le byte 0 des données identifie de manière unique la ressource concernée (voir figure 6.3). La ressource, en l'occurrence la LED RGB, est récupérée via la classe *IOController*. Les données lui sont envoyées. Elle se charge de les décoder. Les composantes rouge (byte 1), verte (byte 2) et bleue (byte 3) sont extraites. Finalement, la commande *write* de la classe *LedRGB* est appelée afin de mettre à jour le duty cycle des trois sorties PWM qui commande la LED RGB.

Le Thread principal appelle périodiquement la méthode *read* afin de traiter les nouvelles données reçues. Dans le cas où l'ID ne correspond à aucune ressource enregistrée, la commande n'est pas traitée.

6.5 Réception des données sur Android

Le scénario inverse est décrit dans le diagramme de séquence ci-dessous. Cette fois-ci, c'est l'accessoire qui transmet l'état de son bouton. Vu par l'application, le bouton est un *BooleanInput*.

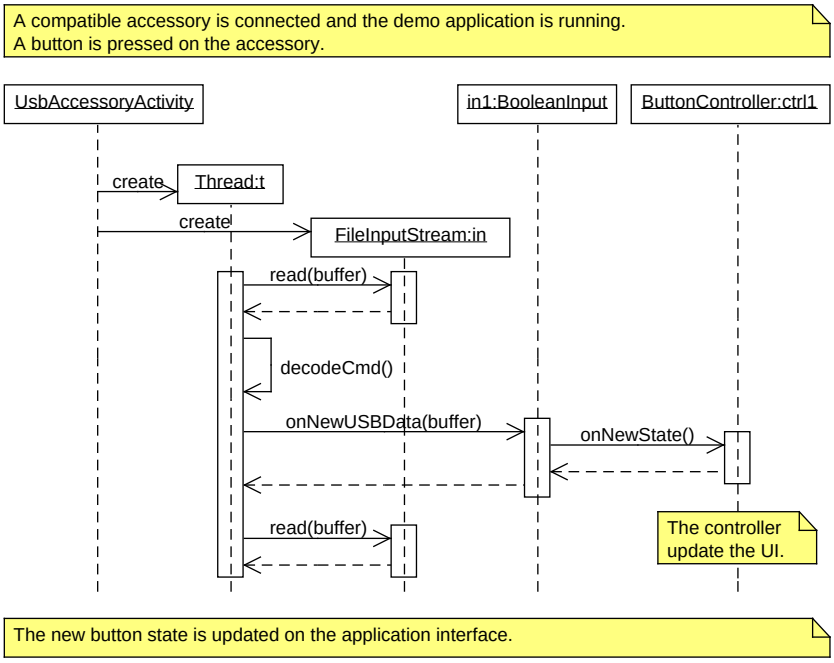


FIGURE 6.6 – Scénario de réception de données sur Android

La classe *BooleanInput* représente une ressource physique. Tous les types de ressources héritent de la classe abstraite *AbstractResource*. Cette classe dispose de la méthode *onNewUsbData* qui permet d'informer à une ressource lorsque des nouvelles données lui sont adressées (voir figure ci-dessus). La méthode inverse *sendUSBData* permet à une ressource de transmettre des données à l'accessoire.

6.6 Utilisation des ressources

Chaque accessoire USB dispose de différentes E/S, de différents types. La librairie *AndroidAccessoy* développée facilite grandement l'accès à ces ressources. La méthode *getResource* permet de récupérer les E/S de l'accessoire de trois manières différentes. Les prototypes de cette fonction sont les suivants :

```
1 public AbstractResource getResource(int index);
2 public AbstractResource getResource(String name);
3 public List<AbstractResource> getResource(ResourceType type);
```

Listing 6.2 – Prototype des fonctions d'accès aux ressources

Voici trois exemples d'utilisation de ces méthodes :

1. Accès à une ressource via son identifiant unique :

```
1 AndroidAccessory board = ...
2 Lcd lcd = (Lcd) board.getResource(9);
3 lcd.print("Hello World", 0, 1);
```

Listing 6.3 – Accès à une ressource via son ID

2. Accès via son nom :

```
1 AndroidAccessory board = ...
2 Timestamp time = (Timestamp) board.getResource("rtc0");
3 time.write(new Date());
```

Listing 6.4 – Accès à une ressource via son nom

3. Accès via un type de ressources particulier :

```
1 AndroidAccessory board = ...
2 List<AbstractResource> list = board.getResource(ResourceType.LED);
3 // Set all LEDs on
4 for (AbstractResource r : list) {
5   BooleanOutput o = (BooleanOutput) r;
6   o.write(true);
7 }
```

Listing 6.5 – Accès à un type de ressources

L'identifiant et le nom des ressources sont uniques, mais plusieurs ressources du même type peuvent exister. C'est pourquoi dans le troisième exemple ci-dessus, la méthode *getResource* retourne une liste de ressources (nombre de ressources variables). Il est intéressant de pouvoir récupérer toutes les ressources d'un même type pour par exemple créer un contrôleur qui gère tous les boutons de l'accessoire et qui peut donc s'adapter au nombre de boutons disponibles. L'exemple ci-dessus permet d'allumer toutes les LEDs disponibles sur l'accessoire via la boucle de la ligne 4.

Le dernier exemple ci-dessous présente comment ajouter un *callback* sur une entrée, dans ce cas un bouton :

```
1 AndroidAccessory board = ...
2 BooleanInput switch0 = (BooleanInput) board.getResource("switch0");
3 switch0.setListener(new OnClickListener() {
4   @Override
5   public void onNewState(BooleanInput input, boolean value) {
6     Log.d(TAG, "New␣state␣" + value);
7   }
8 });
```

Listing 6.6 – Utilisation d'un callback sur une entrée.

Lorsqu'une entrée de l'accessoire change d'état, la nouvelle valeur est automatiquement transmise au périphérique Android et l'interface peut par exemple être mise à jour dans la fonction *onNewState*. Il est aussi possible de fonctionner par polling en lisant la valeur de l'entrée de l'accessoire via la méthode *read*.

Chapitre 7

Applications de démonstration Android

7.1 Introduction

Comme déjà mentionné au début de ce rapport, le protocole Android Accessory est compatible à partir de la version 2.3.4 d'Android, en utilisant une librairie additionnelle pour assurer la compatibilité. Les applications de démonstration développées seront donc compatibles dès l'API 10 d'Android. Les téléphones équipés d'une version moins récente ne pourront tout simplement pas installer les applications qui leur seraient de toute manière inutiles sans un accessoire USB.

Les applications de démonstration sont toutes compatibles avec l'accessoire *uGates-mini*, de même qu'avec l'accessoire USB *uGates* pour l'une d'entre-elles. Les trois applications développées utilisent les différentes ressources à disposition sur les deux accessoires disponibles. Toutes les applications ne sont malheureusement pas compatibles avec l'accessoire *uGates*, puisqu'il ne possède pas de sortie audio. La compatibilité des applications est résumée dans le tableau ci-dessous :

Application	Descrption	Compatibilité	
		uGates	*uGates-mini*
IOController	Pilotage des E/S des accessoires, comme les LEDs, les boutons, l'écran, etc.	✓	✓
SignalGenerator	Générateur de signaux et amplificateur. Nécessite une sortie audio.		✓
WavePlayer	Lecteur de musique pour Android. Nécessite une sortie audio.		✓

Table 7.1 – Compatibilité des applications Android développées

Les trois applications utilisent une librairie commune qui facilite la communication avec l'accessoire et qui permet de détecter automatiquement si ce dernier est compatible ou non. Le développeur a ensuite accès aux E/S de l'accessoire.

Les trois copies d'écran suivantes présentent les écrans communs entre les différentes applications :

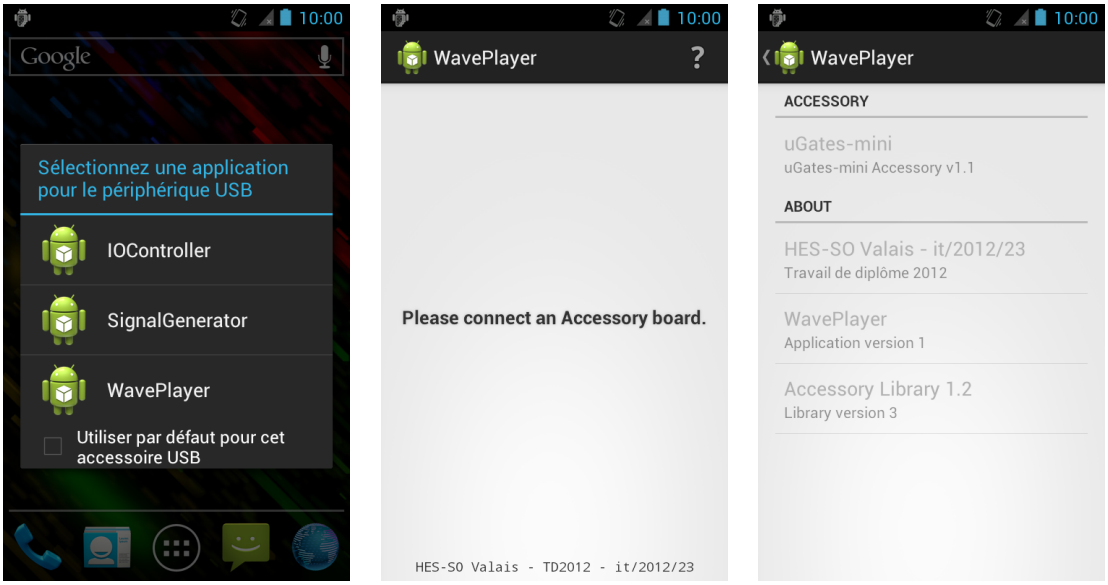


FIGURE 7.1 – Présentation des applications Android

Lorsque l'accessoire *uGates-mini* est connecté, le message de la figure de gauche apparait. Une fenêtre s'affiche avec la liste des applications compatibles avec ce dernier. L'utilisateur sélectionne ensuite l'application qu'il souhaite utiliser. La deuxième capture, au centre, présente l'écran principal des applications. Un message d'erreur apparait si aucun accessoire n'est connecté. Finalement, il est possible d'afficher les informations concernant l'accessoire connecté et l'application, comme le montre la copie d'écran de droite.

Le développement de chaque application est détaillé séparément dans la suite du chapitre. Le code source complet est disponible sur CD-ROM. Certaines parties de code importantes sont annexées.

7.2 Compatibilité

La librairie Open Accessory est utilisée par les applications pour communiquer avec l'accessoire USB. Dès Android 3.1, cette API est supportée nativement. Afin de pouvoir supporter les versions d'Android dès la version 2.3.4, une librairie additionnelle (Google APIs Add-On) via laquelle l'API est accessible doit être inclue dans le projet. Lors de la création de l'application Android, il est important de sélectionner « *Android 2.3.3 + Google APIs* ». La librairie de compatibilité la plus récente peut ensuite être ajoutée automatiquement dans le projet via *Android Tools, Add Support Library*. La figure ci-dessous présente les différentes librairies externes utilisées pour assurer la compatibilité de la librairie Open Accessory dès les téléphones Android 2.3.4 :

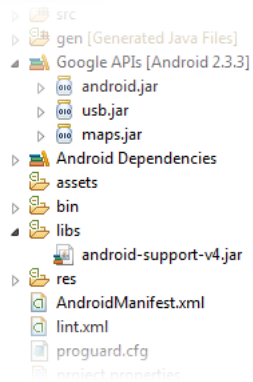


FIGURE 7.2 – Librairies externes Android

A noter que certains appareils peuvent ne pas supporter l'API Open Accessory bien qu'ils aient une version d'Android compatible. En effet, chaque fabricant peut choisir les librairies fournies avec ses téléphones. La ligne de code ci-dessous doit donc impérativement être ajoutée au *manifest* de l'application :

```
1 <uses-feature android:name="android.hardware.usb.accessory" />
```

Ainsi, les téléphones qui ne disposent pas de cette fonctionnalité ne pourront pas installer l'application. De manière générale, toutes les « ressources » qu'utilise une application (ex : caméra, NFC, GPS, etc.) doivent être déclarées dans le manifest par mesure de sécurité (droits d'accès aux ressources) ainsi qu'à titre d'information pour l'utilisateur lors de l'installation de l'application.

7.3 Manifest

Les applications utilisent un hardware spécifique ainsi que des librairies additionnelles. Le manifest de l'application est donc important. Il regroupe toutes ces informations dans un fichier XML présenté ci-dessous :

```
1  <manifest ...>
2    <uses-feature android:name="android.hardware.usb.accessory" />
3    <uses-sdk android:minSdkVersion="10" />
4    ...
5    <application>
6      <uses-library android:name="com.android.future.usb.accessory" />
7      <activity ...>
8        ...
9        <intent-filter>
10         <action android:name="android.hardware.usb.action.
               USB_ACCESSORY_ATTACHED" />
11       </intent-filter>
12       <meta-data android:name="android.hardware.usb.action.
             USB_ACCESSORY_ATTACHED"
13       android:resource="@xml/accessory_filter" />
14     </activity>
15   </application>
16 </manifest>
```

La ligne n°3 spécifie la version minimum du SDK compatible. La ligne 6 indique l'utilisation de la librairie additionnelle. Il faut ensuite déclarer une activité qui sera chargée de la communication avec l'accessoire USB. La ligne 10 permet à l'activité (via un *BroadcastReceiver*) d'être informée lorsqu'un accessoire USB est connecté ou déconnecté. Finalement, il est possible de filtrer les accessoires qui peuvent utiliser l'application. Ce filtre peut être utilisé afin de rentre compatible son application uniquement avec des accessoires d'un certain type, d'une certaine marque, etc. Il est paramétrable à l'aide d'un fichier XML qui est référencé par les lignes 12 et 13 ci-dessus. Dans cet exemple, le fichier se trouve dans « *res/xml/accessory_filter.xml* ».

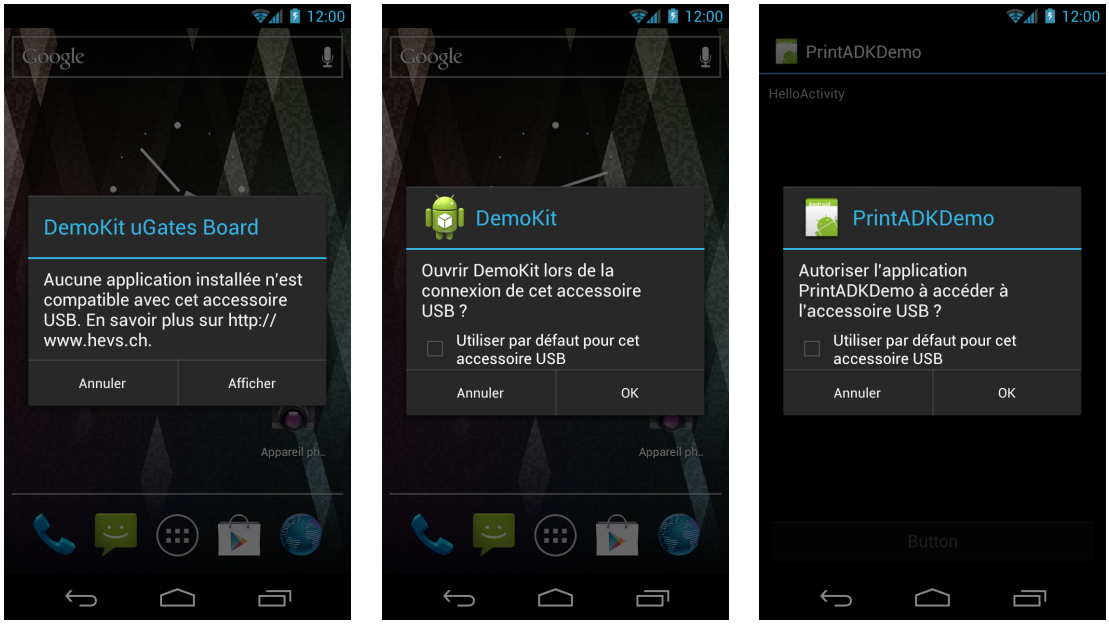
Le fichier complet appelé « fichier de filtre » est disponible ci-dessous :

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3   <usb-accessory manufacturer="HEVs" model="uGates" version="1.0" />
4   <usb-accessory manufacturer="Google" model="DemoKit" version="1.0"/>
5 </resources>
```

La liste des accessoires USB compatibles avec l'application est énumérée dans ce fichier. Il est possible de spécifier le nom du fabricant, le modèle et la version de chaque accessoire. Pour rappel, lorsque l'accessoire se connecte, il transmet les trois informations ci-dessus ainsi qu'un URI et son numéro de série. De cette façon, les accessoires non compatibles peuvent être filtrés. Cette liste de compatibilité permet aussi de démarrer une application compatible avec l'accessoire lors de sa connexion. Le processus de connexion est détaillé dans la suite de ce chapitre.

7.4 Connexion d'un accessoire USB

Lors de la connexion d'un quelconque accessoire USB, plusieurs étapes ont lieu avant de pouvoir communiquer avec ce dernier. Evidemment, par sécurité, l'utilisateur doit donner son accord pour employer l'accessoire. Trois scénarios sont présentés ci-dessous. Ils sont gérés par la librairie Open Accessory. Les captures d'écran ont été prises sur un téléphone Android 4. Selon les versions du système, ces images peuvent légèrement varier :



Scénario 1 Scénario 2 Scénario 3

FIGURE 7.3 – Scénarios de connexion d'un accessoire USB

Ces scénarios sont expliqués ci-dessous :

1. Le premier scénario, à gauche, se produit lorsqu'aucune application n'est compatible avec l'accessoire connecté. Autrement dit, l'accessoire identifié par son fabricant, son modèle et sa version n'est enregistré dans aucune application (via le fichier XML de filtre). Le message d'erreur affiché est prédéfini. Il est possible de visiter l'uri enregistré dans l'accessoire via un browser pour par exemple télécharger une application compatible ou obtenir de plus amples informations.
2. Cette fois-ci, l'accessoire est compatible avec l'application « DemoKit » préalablement installée sur le téléphone. En cliquant sur OK, l'application s'ouvre automatiquement et la communication avec cet accessoire peut être établie. L'utilisateur peut aussi choisir de ne rien faire en annulant l'opération.
3. Dans le dernier scénario, c'est l'utilisateur qui a choisi de démarrer manuellement l'application « *PrintADKDemo* ». Une alerte similaire au deuxième scénario apparait si l'accessoire connecté est compatible. L'utilisateur pourra alors interagir avec ce dernier via l'application. Si l'accessoire n'est pas compatible, c'est comme s'il n'était pas connecté.

7.5 Application générique

Plusieurs accessoires USB et cartes de développement sont disponibles sur le marché (voir chapitre 2.2 page 11). Il est donc indispensable de développer une application générique qui puisse communiquer avec plusieurs d'entre-eux. Ceci peut être très intéressant pour un fabricant qui commercialise plusieurs accessoires par exemple, car une seule application est publiée. De plus, l'utilisateur ne devra installer qu'une seule application.

Pour mettre un place ce système de généricité et de multi-compatibilité, il faut développer un système simple qui permet de lister les accessoires compatibles avec l'application développée. C'est une surcouche qui n'est pas intégrée dans la librairie Open Accessory. Les paramètres de compatibilité des différentes cartes seront enregistrés dans un fichier XML de ressources, très couramment utilisé dans Android. L'exemple d'un tel fichier est présenté ci-dessous. On y retrouve la configuration complète de la carte *uGates-mini* :

```
<?xml version="1.0" encoding="UTF-8"?>
<boards>
  <!-- uGates-mini Board - TD2012 -->
  <AndroidAccessory manufacturer="HES-SO␣Valais"
    model="uGates-mini" version="1.1">
    <!-- 3 LEDs - digital outputs -->
    <DigitalOutput id="0" type="led" name="led0" />
    <DigitalOutput id="1" type="led" name="led1" />
    <DigitalOutput id="2" type="led" name="led2" />

    <!-- 2 Switch - digital inputs -->
    <DigitalInput id="3" type="button" name="switch0" />
    <DigitalInput id="4" type="button" name="switch1" />

    <!-- LED RGB - color output -->
    <LedRGB id="5" type="ledrgb" name="ledrgb0" />

    <!-- UART - stream output -->
    <Uart id="6" type="uart" name="uart0" />

    <!-- RTC - date and time -->
    <Rtc id="7" type="time" name="rtc0" />

    <!-- Analog potentiometer -->
    <Pot id="8" type="pot" name="pot0" />

    <!-- LCD screen -->
    <Lcd id="9" type="lcd" name="lcd0" />

    <!-- Audio output -->
    <Audio id="128" type="audio" name="audio0"
      sampleRate="16000" stereo="false" />
  </AndroidAccessory>

  <!-- HEVs uGates Board -->
  <AndroidAccessory manufacturer="HEVs" model="uGates" version="1.0">
    <!-- ... -->
  </AndroidAccessory>

  <!-- Google DemoKit Board -->
  <AndroidAccessory manufacturer="Google" model="DemoKit" version="1.0">
    <!-- ... -->
  </AndroidAccessory>
</boards>
```

Ce fichier est disponible dans le dossier « *res/xml/boards.xml* » de l'application et référence les accessoires compatibles. Chaque application de démonstration dispose de son propre fichier de compatibilité. Lors de la connexion d'un accessoire, une fonction vérifie si sa configuration est disponible dans le fichier présenté ci-dessus. Si tel n'est pas le cas, un message d'erreur sera affiché pour informer l'utilisateur que l'application n'est pas compatible avec cet accessoire.

Ce processus est résumé dans l'image ci-dessous :

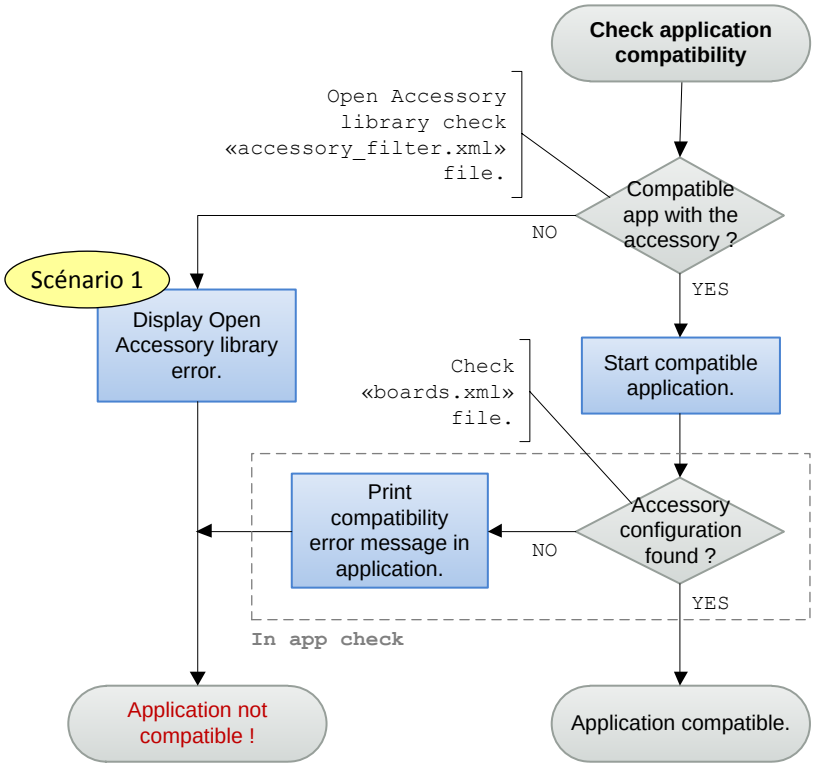


FIGURE 7.4 – Détection d'une application compatible

Afin que cette erreur ne se produise pas, les accessoires doivent être définis à la fois dans le fichier de filtres (voir chapitre 7.3 page 40) et dans le fichier de compatibilité présenté ci-dessus.

7.6 Architecture

Les développements sont divisés en quatre projets Android. Trois projets sont les applications Android de démonstration. Le dernier projet *AccessoryLibrary* est une librairie Android utilisée par les trois autres applications. Elle ne peut pas être directement exécutée.

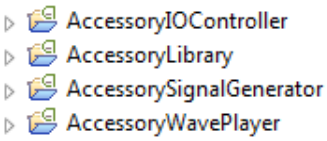


FIGURE 7.5 – Liste des projets Android

Deux fichiers de configuration XML sont nécessaires pour la configuration des applications qui communiquent avec des accessoires :

– *res/xml/accessory_filter.xml*
  Définit les accessoires compatibles avec l'application, dans ce cas, les accessoires *uGates* et *uGates-mini*.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3   <!-- Compatible with uGates and uGates-mini Accessories -->
4   <usb-accessory model="uGates-mini" manufacturer="HEVs" version="1.1"/>
5   <usb-accessory model="uGates" manufacturer="HEVs" version="1.0" />
6 </resources>
```

Listing 7.1 – Exemple d'un fichier de filtres d'accessoires compatibles

– *res/xml/boards.xml*
Définit la configuration et les E/S disponibles pour chaque accessoire compatible avec l'application (voir chapitre 7.5 page 41).

La librairie met à disposition la classe *UsbAccessoryActivity* qui hérite de la classe *Activity* d'Android. Cette classe met à disposition différentes méthodes et attributs pour communiquer facilement avec un accessoire USB. Voici les méthodes mises à disposition par l'interface *UsbAccessoryInterface* :

```
1 // Enable/disable the UI if an accessory is connected or not.
2 public void onEnabledControls(boolean enabled);
3 // Called when a compatible Accessory is connected.
4 public void onAccessoryConnected(AndroidAccessory board);
5 // Called when the Accessory is disconnected.
6 public void onAccessoryDeconnected(AndroidAccessory board);
7 // Called if the connected Accessory is not compatible.
8 public void onAccessoryNotCompatible(UsbAccessory a);
```

Listing 7.2 – Méthodes fournies par la classe UsbAccessoryActivity

Cette classe gère la connexion/déconnexion avec l'accessoire, la vérification de sa compatibilité, l'envoi et la réception des données USB, etc. En héritant de cette classe, les applications peuvent donc facilement utiliser et communiquer avec l'accessoire USB via les différentes méthodes présentées ci-dessus.

7.7 Application IOController

La première application développée permet d'utiliser les entrées/sorties des accessoires. Elle permet de tester l'envoi de données bidirectionnelles entre l'accessoire et le périphérique Android. Deux onglets permettent à l'utilisateur de commander les entrées, et respectivement les sorties, comme le montrent les deux captures de l'application ci-dessous :

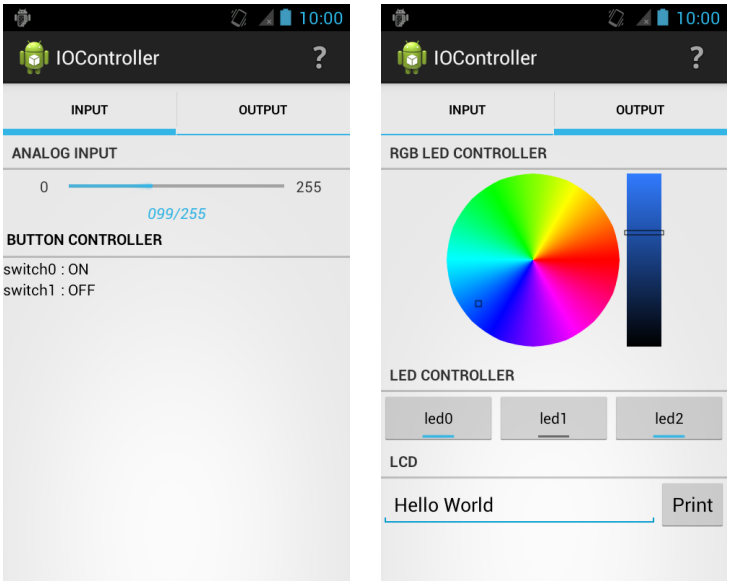


FIGURE 7.6 – Captures de l'application IOControl

Pour cet exemple, c'est l'accessoire *uGates-mini* qui est connecté. L'entrée analogique, et l'écran LCD ne sont pas disponibles sur la carte *uGates*. L'application s'adapte au type d'accessoire connecté, l'interface n'est donc pas identique pour les deux kits.

Lorsque l'accessoire *uGates-mini* est connecté, son horloge est synchronisée avec celle du téléphone. Le nom de l'application ainsi que le modèle du téléphone sont inscrits sur l'écran LCD, comme le montre l'image suivante (connexion avec le téléphone Nexus S de Google) :

FIGURE 7.7 – IOControl - écran LCD

Dans l'onglet « Input », on peut visualiser la valeur du potentiomètre ainsi que l'état ON ou OFF des deux boutons. Lorsque la valeur d'une entrée change, l'accessoire envoie automatiquement la nouvelle valeur vers le périphérique Android (*event-driven*) et l'interface de l'application est mise à jour. Chaque entrée dispose d'une fonction *callback* qui est appelée lorsqu'un changement d'état a lieu.

Trois contrôleurs permettent à l'utilisateur de commander les sorties dans l'onglet « Output ». Tout d'abord, un sélecteur [1] permet de modifier la couleur de la LED RGB, ainsi que son intensité. Les trois LED vertes de l'accessoire peuvent être commandées séparément via le deuxième contrôleur. Finalement, il est encore possible d'afficher un texte sur l'écran de l'accessoire *uGates-mini*. La position du texte à l'écran est fixe.

7.7.1 Thread de réception des données

Sur Android, un Thread dédié permet de réceptionner les données envoyées par USB. Les données reçues correspondent par exemple à l'état des boutons de l'accessoire, ou encore à la valeur du potentiomètre. Pour rappel, l'état des entrées de l'accessoire n'est pas envoyé périodiquement, mais uniquement lorsque des changements d'état ont lieu.

Le Thread de réception des données est géré par la classe *UsbAccessoryActivity* de la librairie *AccessoryLibrary*. Voici comment les données sont réceptionnées et décodées :

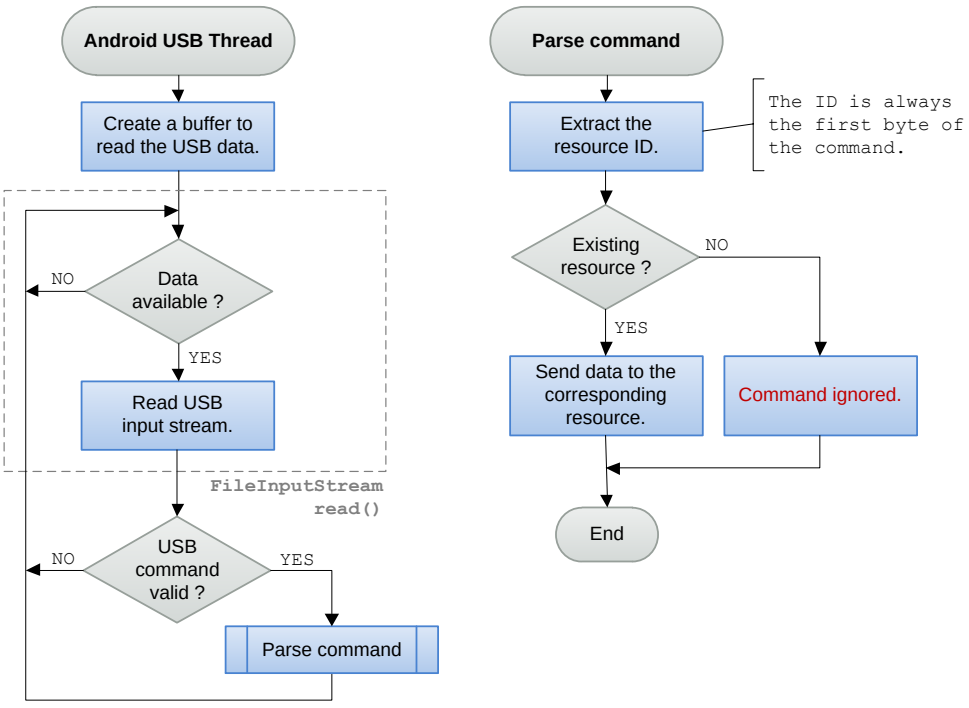


FIGURE 7.8 – Procédure de réception des données USB sur Android

Le stream d'entrée est de type *FileInputStream*. Après avoir créé un buffer local, la méthode read du stream permet de lire les données USB. Le nombre de byte lu est retourné par cette méthode. Une fois la validité de la trame vérifiée, la commande peut être décodée. Le byte 0 indique l'ID de la ressource concernée. Si elle existe, les données lui sont envoyées via la fonction *onNewUSBData* (voir chapitre

1. SuperdryColorPicker - http ://code.google.com/p/superdry-colorpicker/

6.5). Les commandes sont ignorées si elles ne sont pas valides ou si l'identifiant ne correspond à aucune ressource. Le code du Thread de réception est disponible en annexe J.

Fermeture de la connexion avec l'accessoire

Les données USB sont lues en boucle par le Thread. La méthode *read* est blocante. Lorsque des données sont reçues, le nombre de byte lu est retourné. Une exception a lieu lorsque le stream est fermé.

Lorsque l'accessoire est déconnecté ou l'application fermée, le Thread de réception doit être arrêté et le stream USB fermé. Il est important de libérer l'accès à l'accessoire pour permettre par exemple à une autre application de l'utiliser. L'utilisation du Thread et la méthode *read* bloquante rendent ces opérations difficiles.

Voici l'erreur qui se produit à la réouverture de l'application :

```
1 E/UsbDeviceManagerJNI(258): could not open /dev/usb_accessory
```

Listing 7.3 – Erreur du driver USB sur Android

Cette erreur provient du driver C++ USB et indique qu'il est impossible d'ouvrir une connexion avec l'accessoire. Cela signifie que l'ancienne connexion avec l'application est toujours active, car la méthode *read* du stream bloque la fermeture du Thread. Ce problème n'a pas lieu pour la première ouverture de l'application, mais uniquement si cette dernière est fermée puis réouverte.

A ce stade, la solution la plus sûre pour fermer correctement la connexion à l'accessoire est de le débrancher physiquement du téléphone. Le problème rencontré est « connu » et pourrait provenir de la librairie de compatibilité utilisée pour le support des téléphones dès Android 2.3.4. Plusieurs problèmes ont été rapportés à ce sujet sur le site *StackOverflow*. Une *issue*[2] Android est aussi disponible.

Une solution possible consiste à faire un « *soft close* ». L'accessoire doit être informé de la fermeture de l'application et doit envoyer des données (*dummy bytes*) à ce moment là, ce qui permet de débloquer la méthode *read*. La connexion avec l'accessoire peut alors être fermée correctement. Cette solution n'a pas encore pu être testée. Des tests supplémentaires doivent aussi être réalisés pour savoir si le problème provient uniquement de la librairie de compatibilité. Afin d'éviter ce problème, les applications qui ne reçoivent pas de données de l'accessoire ne doivent pas activer le Thread de réception USB.

7.7.2 Commande de la LED RGB

La LED RGB est commandée à l'aide de trois sorties PWM. Chacune d'elles commande une des trois couleurs primaires, soit le rouge (*Red*), le vert (*Green*) et le bleu (*Blue*), comme le montre la capture d'oscilloscope ci-dessous :

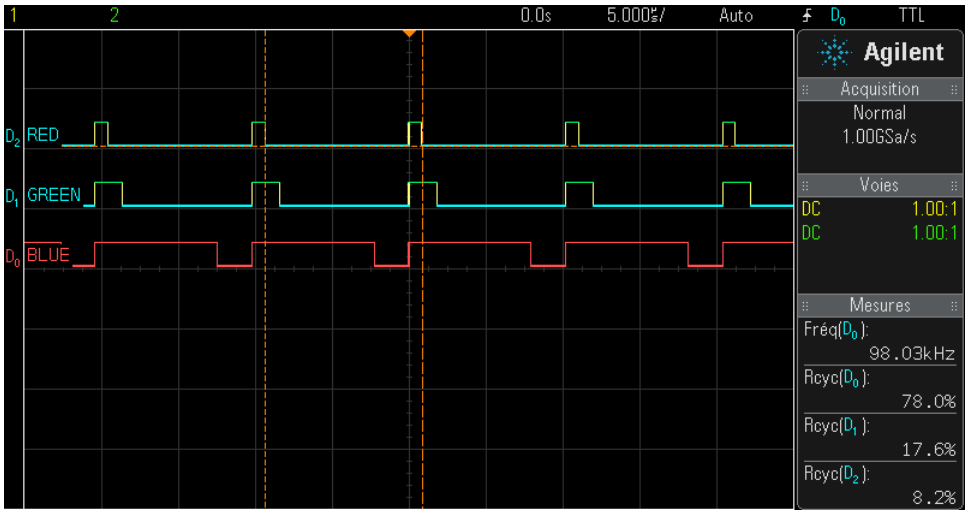


FIGURE 7.9 – Contrôle de la LED RGB par PWM

2. http ://code.google.com/p/android/issues/detail ?id=20545

La fréquence de toutes les sorties PWM est identique. La variation du duty cycle de chaque composante permet de varier la couleur finale de la LED. Pour cet exemple, 78% de bleu , 18% de vert et 8% de rouge, ce qui correspond à une couleur bleutée (*#142EC7*).

7.8 Application SignalGenerator

Cette deuxième application est compatible uniquement avec la carte *uGates-mini*, car elle nécessite une sortie audio. Dans un premier temps, la sortie audio est utilisée pour générer différents signaux, qui peuvent être paramétrés depuis l'application Android que voici :

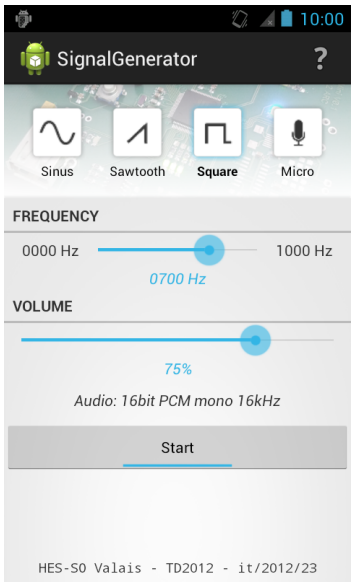


FIGURE 7.10 – Capture de l'application SignalGenerator

Le layout de l'application est inspiré de l'application « Local » de Google Maps. Quatre boutons permettent de sélectionner le type de signal de sortie. Il est possible de générer un sinus, un signal en dents de scie ainsi qu'un signal carré.

La fréquence des signaux est paramétrable et peut varier entre 0 et 1kHz. Afin d'obtenir des signaux « propres » en sortie (sortie audio 16kHz), la fréquence maximale des signaux générés a été limitée à 1kHz.

Cette plage de fréquence est parfaite pour la démonstration, les signaux peuvent être écoutés.

La deuxième *SeekBar* permet de régler l'amplitude (le volume) des signaux générés. La sortie audio peut aussi être activée ou désactivée via le bouton Start/Stop.

Enfin, l'utilisateur peut aussi sélectionner le microphone comme source sonore. Le téléphone peut alors être utilisé comme « mégaphone ». Les sons enregistrés par le micro sont redirigés vers la sortie audio de l'accessoire *uGates-mini*.

L'écran LCD de l'accessoire permet d'afficher le nom de l'application, le type de signal de sortie et sa fréquence, comme le montre la figure ci-dessous :

FIGURE 7.11 – SignalGenerator- écran LCD

7.8.1 Génération des signaux

La fréquence et l'amplitude des signaux générés peuvent être paramétrées par l'utilisateur. Les valeurs des signaux sont des nombres signés codés sur 16 bits. Le volume est compris en 0 et 1.0 et est multiplié à la valeur de sortie.

La fréquence des signaux générés est comprise entre 0 et 1000Hz et la sortie audio est à 16kHz. Bien que les signaux générés soient périodiques, le signal est envoyé comme un stream. En effet, la fréquence de la sortie audio n'est pas forcément un multiple de la fréquence du signal généré. Il n'est donc pas possible d'envoyer uniquement une période du signal, car la taille du *circular buffer* utilisé comme buffer audio sur l'accessoire est fixe. L'implémentation de ce buffer est décrite dans le chapitre 7.9.7.

La méthode utilisée pour générer le signal audio de sortie est illustrée à l'aide de la figure ci-dessous :

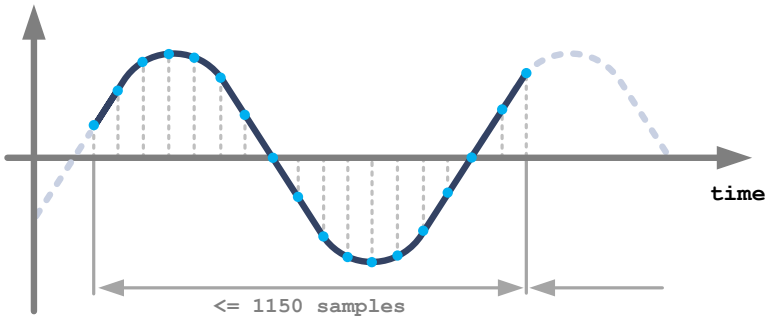


FIGURE 7.12 – Exemple de génération d'un sinus

Prenons l'exemple d'un sinus de fréquence 500Hz. Un certain nombre de samples ont déjà été envoyés vers l'accessoire. La dernière valeur envoyée qui correspond au premier point bleu sur la figure ci-dessus est sauvegardée. Le signal est échantillonné à 16kHz, fréquence de la sortie audio. Le téléphone calcule ensuite les N samples suivants. Les valeurs calculées sont représentées par les points bleus. Le nombre N de samples à fournir est calculé dynamiquement par le Thread audio et est limité à 1150 samples. Plus de détails sont disponibles dans le prochain chapitre. Le dernier échantillon calculé est sauvé et permettra de connaître la valeur du prochain sample. La même idée est appliquée pour la génération des autres types de signaux.

Selon le théorème d'échantillonnage, la fréquence des signaux générés ne doit pas dépasser 8kHz. Cette fréquence limite est liée aux paramètres de la sortie audio. Pour garder des signaux qui restent visuellement « propres », surtout dans le cas d'un signal carré, la fréquence maximale des signaux générés a été limitée, avec beaucoup de marge, à 1kHz. Cette limitation ne pose pas de problème pour cette application de démonstration, dont le but n'est pas de générer des signaux de fréquence maximale.

7.8.2 Mesure des signaux générés

Les signaux générés ont été mesurés sur la sortie audio de l'accessoire. Le chapitre suivant décrit comment les données sont envoyées puis traitées par l'accessoire. Voici la mesure des signaux générés, regroupés sur la même capture d'oscilloscope :

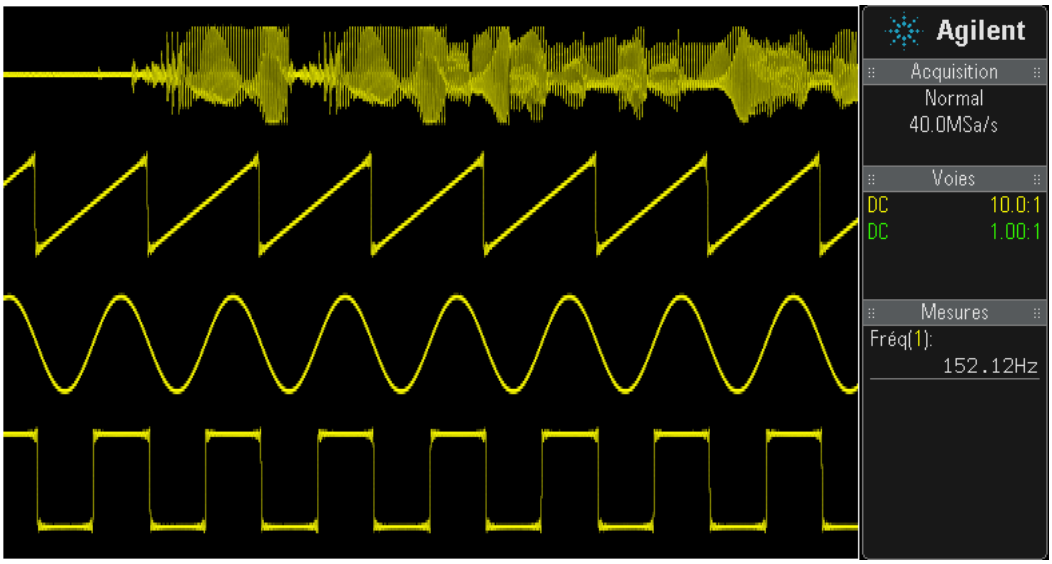


FIGURE 7.13 – Capture des signaux générés

On retrouve le signal de sortie du microphone, le signal en dents de scie, le sinus et finalement le signal carré. Dans cet exemple, le volume a été fixé à 50% et la fréquence à 150Hz pour tous les signaux générés.

Le tableau ci-dessous présente les erreurs des trois types de signaux générés pour différentes valeurs de fréquences :

SIGNAL	FRÉQUENCE	MESURE	ERREUR
Sinus	100 Hz	0101.57 Hz	1.57%
	0500 Hz	0508.44 Hz	1.68%
	1000 Hz	1017.30 Hz	1.73%
Sawtooth	100 Hz	0101.73 Hz	1.73%
	0500 Hz	0508.64 Hz	1.73%
	1000 Hz	1017.50 Hz	1.75%
Square	100 Hz	0101.73 Hz	1.73%
	0500 Hz	0508.60 Hz	1.72%
	1000 Hz	1017.00 Hz	1.70%

TABLE 7.2 – Mesures de précision des signaux générés

Le but de cette application de démonstration n'est pas d'obtenir le signal le plus précis possible. L'erreur mesurée reste faible, elle se situe entre 1.5 et 2 %. Entre 100 et 1000Hz, elle est indépendante de la fréquence et du type de signal généré. La fréquence d'envoi des trames audio depuis Android est légèrement plus rapide que la fréquence réelle de la sortie audio. Ceci explique la légère différence de fréquence mesurée.

7.9 Application WavePlayer

Cette dernière application de démonstration permet à l'utilisateur, via un lecteur audio, d'écouter de la musique stockée sur son téléphone et de la diffuser sur la sortie audio amplifiée de la docking station.

Elle se présente sous cette forme :

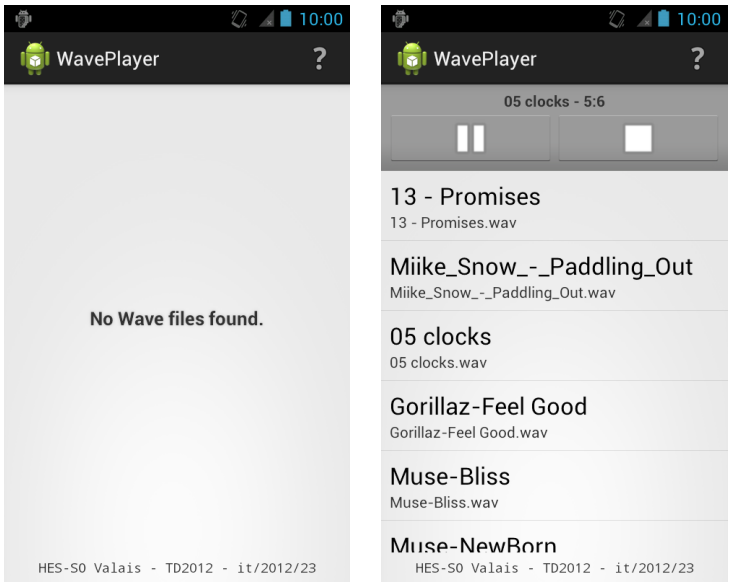


FIGURE 7.14 – Capture de l'application WavePlayer

Les musiques compatibles avec la sortie audio de l'accessoire sont affichées dans la liste. Si aucun fichier audio n'a été trouvé, un message d'erreur est affiché, comme le montre la figure de gauche ci-dessus.

Dans la liste principale, on y retrouve le titre et le nom du fichier audio. En cliquant sur un des titres, la musique correspondante est jouée sur la sortie audio de l'accessoire *uGates-mini*. Il est possible de changer le titre en cours de lecture à tout moment. Lorsqu'une chanson est en cours de lecture, deux boutons permettent de mettre en pause ou de stopper la lecture. La durée du fichier audio est aussi affichée.

Le titre de la chanson en cours de lecture, la durée totale de celle-ci ainsi que le nom de l'application sont affichés sur l'écran de l'accessoire. En voici un aperçu :

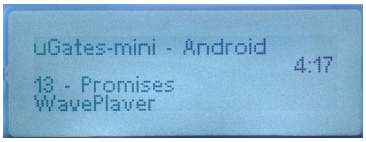


FIGURE 7.15 – WavePlayer- écran LCD

Le volume de la musique peut être réglé directement sur la docking-station à l'aide du potentiomètre de la carte d'extension.

7.9.1 Audio sur Android

Nous allons nous intéresser aux différentes fonctionnalités disponibles sur Android pour utiliser l'audio et travailler avec des sons. Les aspects de la vidéo ne seront pas abordés. Les classes principales du package *android.media* sont décrites dans le tableau ci-dessous :

Nom	Description
android.media.MediaPlayer	Lecteur générique pour de l'audio ou de la vidéo. Exemple : lecture d'un fichier audio local ou en streaming. Un grand nombre de formats audio peuvent être décodés.
android.media.audiofx.Visualizer	Permet de récupérer le flux audio à des fins de visualisation. Exemple : affichage de l'amplitude ou du spectre fréquentiel audio.
android.media.MediaRecorder	Permet d'enregistrer de l'audio ou de la vidéo. Exemple : enregistrement du micro et création d'un fichier audio.
android.media.AudioTrack	Permet de jouer une piste audio brute à l'aide d'un flux PCM brute.
android.media.AudioManager	Gestionnaire de volume et de sonnerie du téléphone. Permet d'enclencher le micro, de régler le volume des haut-parleurs, etc.
android.media.AudioRecord	Cette classe permet d'enregistrer de l'audio depuis différentes sources, dont le micro du téléphone.

Table 7.3 – Classes audio sur Android

La classe *MediaPlayer* est la plus standard. Elle est disponible sur tous les OS et permet de décoder des sons et des vidéos. Les sources sonores peuvent être multiples et provenir directement du micro du téléphone, d'un fichier audio ou d'un flux (radio en streaming). Un grand nombre de formats audio sont pris en charge [3] et décodés par le *MediaPlayer*.

La classe *MediaRecorder* permet d'enregistrer une source audio (exemple microphone) dans un fichier standardisé, qui pourra être rejoué plus tard par le *MediaPlayer*. Le volume des haut-parleurs du téléphone peut être réglé via la classe *AudioManager* qui donne accès à un service de l'OS. Cette classe permet aussi d'activer ou de désactiver le microphone. Enfin, la classe *AudioTrack* peut être utilisée pour décoder des flux audio PCM qui seraient par exemple reçus en streaming.

On remarque donc qu'à l'aide des différentes classes présentées ci-dessus, tout est fait pour lire facilement des fichiers multimédia, pour jouer des sons, effets sonores, etc. Il est cependant plus complexe de réaliser l'opération inverse. Seule la classe *Visualizer* permet de récupérer un flux audio. Cette classe fait partie du package *media.audiofx*. Comme son nom l'indique, elle permet de récupérer l'audio en cours de lecture, d'un *MediaPlayer* ou d'un *AudioTrack*, ceci à des fins de visualisation. Elle possède deux fonctions principales :

1. Récupérer la forme d'onde du stream audio joué. La fonction *Visualizer.getWaveForm()* retourne un tableau de bytes qui représente des samples PCM 8 bits, mono, non signés.
2. La fonction Visualizer.*getFft()* effectue une FFT [4] simple et retourne une capture spectrale du son.

La fonction *getWaveForm()* fournit uniquement des samples PCM 8 bits. Aucune autre configuration n'est possible. Cette classe convient donc parfaitement pour la réalisation d'un vu-mètre ou d'un graphique spectral par exemple. Les samples PCM sont de trop mauvaise qualité pour être utilisés. C'est sans doute par choix de performances que Google ne permet pas d'augmenter la qualité de ces samples.

7.9.1.1 Microphone

Il faut encore mentionner le microphone des téléphones. Ce dernier est une ressource audio vue comme un capteur qui peut être activé ou désactivé. La classe *AudioRecord* permet de lire facilement les sons enregistrés par le micro. L'encodage des samples fournis par le micro peut être paramétré, comme le montre l'exemple de code suivant :

3. Liste des formats audio et vidéo supportés - http ://developer.android.com/guide/appendix/media-formats.html
4. Fast Fourier transform : http ://en.wikipedia.org/wiki/Fast_Fourier_transform

```
1  int bufferSize = AudioRecord.getMinBufferSize(16000, AudioFormat.CHANNEL_IN_MONO,
2        AudioFormat.ENCODING_PCM_16BIT);
3
4  byte[] buffer = new buffer[bufferSize]; // Audio buffer in bytes
5
6  // Configure audio output
7  AudioRecord recorder = new AudioRecord(AudioSource.MIC, 16000,
8        AudioFormat.CHANNEL_IN_MONO, AudioFormat.ENCODING_PCM_16BIT, bufferSize);
9
10 recorder.startRecording();
11
12 // Audio loop
13 while(...) {
14   // Read the microphone audio stream
15   int n = recorder.read(buffer, 0, buffer.length);
16   ...
17 }
```

Listing 7.4 – Encodage des samples audio du micro.

Tout d'abord, en ligne 1, la méthode *getMinBufferSize* calcule le nombre minimal de bytes nécessaires pour stocker les samples du micro. La taille du buffer dépend bien sûr de la qualité audio choisie.

La classe *AudioRecord* peut ensuite être instanciée en spécifiant les paramètres audio (ligne 7). Dans cet exemple, la source audio d'entrée est le micro. Le flux audio est un flux mono, encodé en PCM 16 bits à 16kHz.

Finalement, la méthode *read* permet de lire les samples du micro, selon la configuration choisie. Les samples sont sauvés dans un buffer. La méthode read permet de récupérer des samples 8 ou 16 bits, en fonction du type de buffer déclaré. Dans cet exemple, les sample PCM 16 bits sont décomposés en 2 bytes (little indian). La lecture du stream audio du micro doit être réalisée dans un *Thread*. Le microphone peut être désactivé via la méthode *setMicrophoneMute* de la classe *AudioManager*. A noter que l'enregistrement du micro nécessite la permission *android.permission.RECORD_AUDIO*.

7.9.1.2 Synthèse

Aucune fonction de l'API Android ou de service de l'OS ne permettent de récupérer un flux audio de qualité. Pour pouvoir tout de même utiliser la sortie audio de l'accessoire *uGates-mini*, plusieurs solutions sont envisageables.

La première solution consiste à ne pas utiliser le décodeur audio d'Android, mais plutôt son propre décodeur audio. Des décodeurs audio « génériques » et open-source sont disponibles sur le marché. Une fois porté sur Android, un tel décodeur permettrait de lire des fichiers audio stockés sur le téléphone, de les décoder selon les paramètres audio choisis, et finalement le stream audio pourrait être envoyé vers l'accessoire USB. La librairie *MAD*[5] est un décodeur audio qui pourrait par exemple être utilisé. Il permet de décoder des fichiers MP3 et fournit un stream PCM 24 bits en sortie. Le portage d'une telle librairie sur Android nécessite l'utilisation du NDK et de JNI. Un article intéressant[8] présente quelques éléments concernant le portage de *libmad* sur Android.

La deuxième solution, plus simple, consiste à décoder manuellement un fichier audio de type prédéfini, comme les fichiers Wave. En effet, le format audio Wave est un format non-compressé. Les samples audio peuvent donc être facilement extraits, sans algorithmes particuliers. La taille de ces fichiers non-compressés peut s'avérer assez importante. De ce fait, les fichiers audio seront stockés sur la carte SD du téléphone.

La deuxième solution ne permet pas de lire tous les types de fichiers audio, mais elle est plus simple à mettre en œuvre. N'ayant jamais travaillé avec le NDK d'Android, le portage de *libmad* peut s'avérer trop long à réaliser dans le temps imparti. C'est pourquoi la deuxième solution a été choisie. Malheureusement, seul un format audio Wave de type prédéfini pourra être décodé. Cette solution a l'avantage de pouvoir déjà tester le hardware de l'accessoire et de pouvoir tester le Thread audio qui est chargé d'envoyer les samples audio vers l'accessoire.

5. MAD (MPEG Audio Decoder) - http ://sourceforge.net/projects/mad/

7.9.2 Lecture de fichiers Wave

Les fichiers audio Wave sont des fichiers audio non compressés. La lecture des samples qui composent le fichier est donc triviale. C'est pour cela que ce type de fichier a été choisi. Le format des fichiers Wave est bien documenté[16]. De manière simplifiée, voici la structure d'un tel fichier :

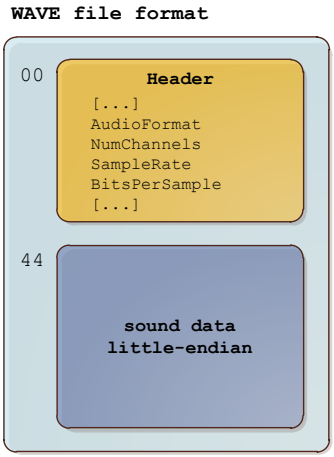


FIGURE 7.16 – Format Wave simplifié

Les fichiers Wave supportent plusieurs types d'encodage. Ces paramètres sont disponibles dans le header. A l'ouverture du fichier, sa compatibilité est vérifiée en fonction des paramètres suivants :

OFFSET (BYTE)	TAILLE (BYTE)	PARAMÈTRE	VALEUR	DESCRIPTION
20	2	*AudioFormat*	**1**	PCM audio data (pas de compression)
22	2	*NumChannels*	**1**	Audio mono
24	4	*SampleRate*	**16000**	16000 kHz
34	2	*BitsPerSample*	**16**	16 bits par sample

TABLE 7.4 – Encodage des fichiers Wave

Si ces paramètres ne correspondent pas, le fichier audio est considéré comme incompatible.

7.9.3 Export de fichiers audio compatibles

Afin d'obtenir des fichiers audio Wave compatibles, le logiciel Audacity [6] a été utilisé. Il permet de convertir une large gamme de fichiers et de formats audio, comme les fichiers mp3 par exemple. La liste complète des formats supportés est disponible sur le site officiel.

Une fois la source audio importée, les pistes stéréo doivent d'abord être séparées en deux pistes mono via l'option « Séparer stéréo vers mono ». Une des pistes doit ensuite être supprimée, car le fichier de Wave de sortie doit être mono. Les paramètres d'exportation du Wave sont :

- format Wave
- 16 bits PCM - signé
- 16000Hz - mono

6. Enregistrement et édition de sons - http ://audacity.sourceforge.net/

Les captures ci-dessous proviennent du logiciel Audacity. On retrouve à gauche le format d'exportation correct et à droite l'ouverture d'un fichier Wave compatible.

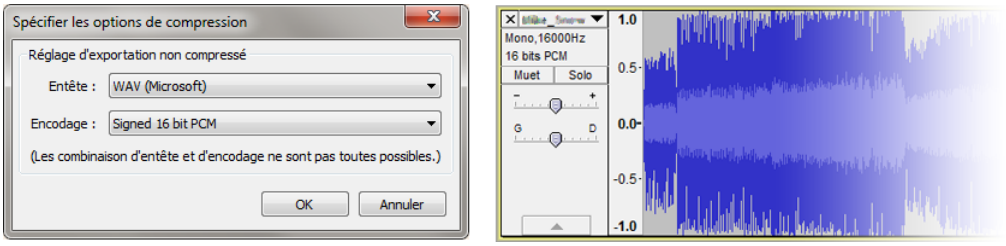


FIGURE 7.17 – Encodage d'un fichier Wave avec Audacity

7.9.4 Récupération des fichiers audio

Tous les fichiers audio compatibles avec la sortie audio de l'accessoire sont affichés dans la liste de l'application. Les fichiers audio ne sont pas stockés dans l'application, mais directement sur le téléphone (sur la carte SD par exemple). L'utilisateur pourra ainsi utiliser ses propres fichiers audio et en ajouter facilement. Il pourra aussi les utiliser avec n'importe quelle autre application audio.

La classe *MediaStore* permet de récupérer des fichiers multimédias stockés sur la mémoire interne comme externe du téléphone, comme le montre le code suivant. Ce code ne nécessite aucune permission sur Android.

```
1  // Informations about a song
2  String[] proj = { MediaStore.Audio.Media._ID, // song id
3    MediaStore.Audio.Media.DATA,  // song path
4    MediaStore.Audio.Media.DISPLAY_NAME,  // song name
5    MediaStore.Audio.Media.SIZE,  // song size
6    MediaStore.Audio.Media.TITLE, // song title
7    MediaStore.Audio.Media.DURATION // song duration [ms]
8  };
9  // Select audio files with a .wav extension
10 String select = MediaStore.Audio.Media.DATA + "␣like␣'%.wav'";
11 // Get all audio files with an SQL request
12 mCursor = managedQuery(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
13   proj, select, null, null);
```

Listing 7.5 – Récupération des fichiers audio sur Android

La méthode *managedQuery* de la classe *Activity* est utilisée pour ne récupérer que les fichiers audio de type « .wav ». Plusieurs informations concernant le fichier audio sont sauvées, comme son nom, son emplacement, sa durée, etc (ligne 2). Ce premier filtre est réalisé à l'aide de la requête *SQL* de la ligne 10.

Le *Cursor* retourné par la méthode *managedQuery* est ensuite mis en forme dans la liste à l'aide d'une *SimpleCursorAdapter*. Un message est affiché si aucun fichier audio n'a été trouvé.

7.9.5 Compatibilité des fichiers audio

Le premier filtre appliqué sur le type et l'extension des fichiers audio n'est pas suffisant. Différents encodages sont disponibles pour les fichiers Wave et seul un type est compatible avec la sortie audio de l'accessoire.

Avant de jouer le fichier audio, à son ouverture donc, le header du fichier Wave est analysé. En cas de non-compatibilité, un message d'erreur est affiché et le fichier ne sera pas joué. Le test de compatibilité est réalisé à l'aide de la classe *WaveHeader*, comme montré dans le code suivant (simplifié) :

```
1  // Open the selected Wave with the file path
2  InputStream in = ...;
3
4  // Read the Wave header
5  WaveHeader header = new WaveHeader();
6  header.read(in);
7
8  // Check the file compatibility
9  if (header.getFormat() != FORMAT_PCM ||
10   header.getNumChannels() != 1 ||
11   header.getSampleRate() != 16000) {
12   // Wave format not compatible. An error message is displayed.
13 }
```

Listing 7.6 – Test de compatibilité des fichiers Wave

7.9.6 Trames audio USB

Les données du fichier Wave sont lues et transmises par paquet. Ces derniers sont envoyés à intervalle régulier et peuvent contenir un nombre de samples variables, comme le montre la figure ci-dessous :

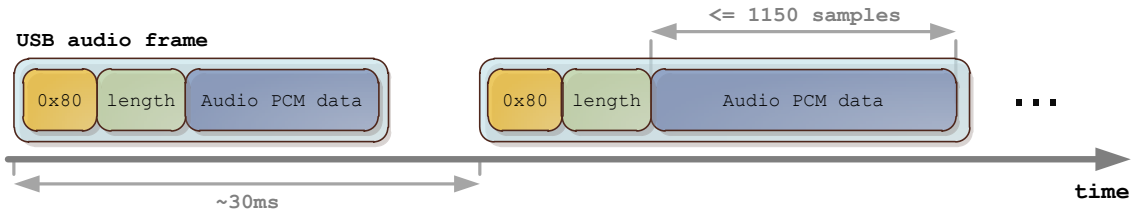


FIGURE 7.18 – USB audio frames

La sortie audio mono est identifiée par l'ID 128. Les trames audio sont transmises toutes les 30ms environ par le *Thread* dédié de l'application Android. Il est important de respecter un délai entre l'envoi de deux trames, afin qu'elles puissent être décodées séparément. Ce délai est réalisé avec la fonction *Thread.sleep* de Java. Selon les mesures réalisées, ce délai peut varier, par exemple lorsque la charge du processeur est élevée. C'est pourquoi le nombre d'échantillons de chaque trame doit être variable.

Il est possible de transmettre jusqu'à 2300 bytes de données audio par trames, soit 1150 samples (mono, PCM 16 bits). Le débit théorique maximum qui peut être délivré par l'application est alors de :

Algorithme 7.1 Calcul du débit audio maximum

$$D_{MAX} = \frac{1150\,samples}{30\,ms} = 38.34\,\text{kSamples/s}$$
$$k = \frac{38.34k}{16k} = 2.4$$

Ce débit maximal théorique est presque 2.5 fois plus rapide que la sortie audio à 16kHz de l'accessoire. La taille et la fréquence d'envoi des trames audio permettent donc d'avoir suffisamment de marge en cas de ralentissement du côté d'Android. Cette marge doit être suffisante afin d'éviter à tout prix un phénomène de famine (*starving*), qui correspondrait à un manque d'échantillons audio sur la docking station. Aussi, pour éviter ce phénomène, la fréquence de la sortie audio a été légèrement accélérée dans le *Thread* audio, ce qui permet de garantir l'arrivée suffisante de nouveaux échantillons sur l'accessoire. Dans le cas où le buffer de réception est plein, le surplus d'échantillons est ignoré.

La taille optimale des trames audio est calculée par le *Thread* audio avant chaque envoi à l'aide de la fonction *System.nanoTime()*. Le nombre moyen de samples envoyés peut être calculé de la manière suivante :

Algorithme 7.2 Nombre moyen de samples audio envoyés par USB

$$Nsamples_{MOY} = 16kHz \cdot 30ms = \frac{2300\,bytes}{k} = 480\,samples = 960\,byte$$

Le nombre de samples contenu dans chaque frame fait partie du header. Le code du *Thread* audio est disponible en annexe K.

7.9.7 Circular audio buffer

Du côté de l'accessoire, la réception des données audio a été implémentée à l'aide d'un *circular buffer*. Ce buffer permet de stocker 3000 samples 32 bits stéréo. L'implémentation du circular buffer est illustrée à l'aide de la figure suivante :

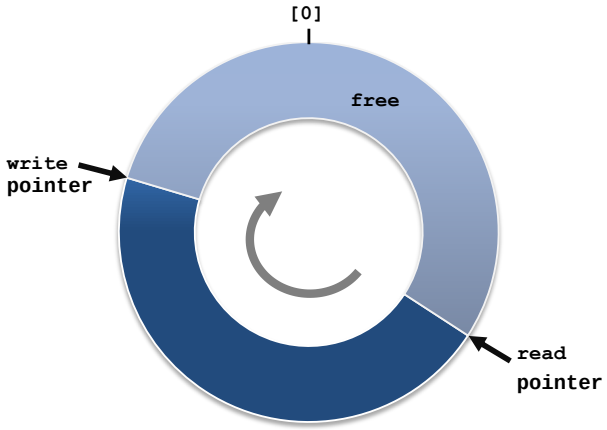


FIGURE 7.19 – Audio circular buffer

La zone en bleu foncé ci-dessus représente les données audio non lues, la zone en bleu clair la zone libre. Deux pointeurs sont utilisés pour respectivement écrire et lire des données. La zone de mémoire libre se situe donc entre ces deux pointeurs (*read* - *write*). Lorsque des nouvelles données arrivent par USB, elles sont copiées dans ce buffer audio et le pointeur d'écriture est incrémenté ou rebouclé. Les anciennes données sont automatiquement écrasées.

Lorsque le buffer est plein, 3000 samples par canaux sont stockés. Voici le temps que met le buffer pour se vider complètement :

Algorithme 7.3 Temps pour vider complètement le buffer audio

$$Tbuffer = \frac{3000\,samples}{16\,kSamples/s} = 187.5ms$$

Après ce temps, toutes les données du buffer audio auront été lues.

Le contrôleur *GPDMA* (General Purpose DMA) du processeur est configuré pour envoyer les samples audio contenus dans le buffer vers la FIFO de sortie du contrôleur I2S. Un codec audio décode ensuite les données transmises par I2S. La copie des données est automatisée par le contrôleur DMA hardware, aucun traitement ne peut donc être effectué sur les samples audio. Ces derniers doivent être stockés selon le format requis par le contrôleur I2S, illustré dans la figure ci-dessous :

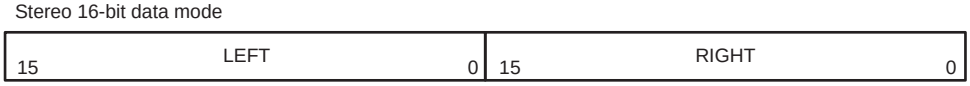


FIGURE 7.20 – Format des données du buffer audio

La figure ci-dessus est extraite du datasheet du processeur *LPC1758*. D'autres configurations sont disponibles : mono/stéréo, 8/16 ou 32 bits. La sortie audio de la carte *uGates-mini* est configurée en stéréo 16 bits, 16kHz.

Le contrôleur DMA est paramétré pour copier toutes les données contenues dans le buffer audio vers la FIFO I2S. Il s'agit donc d'un transfert de mémoire à périphérique (*M2P - Memory to peripheral*). Une interruption DMA a lieu lorsque la copie des données est terminée. Le nouveau transfert doit être démarré manuellement. Le contenu du circular buffer est copié en boucle vers la sortie I2S. Le codec audio dispose d'une entrée *mute* (GPIO) qui permet de la désactiver la sortie audio.

Le code source de la classe *RingBuffer* est disponible avec le code source de l'accessoire *uGates-mini* sur CD-ROM.

Chapitre 8

Synthèse

8.1 Tests

Des tests individuels ont été réalisés pour chaque partie des développements. A ce jour, l'accessoire USB *uGates-mini* dispose des fonctions nécessaires pour communiquer avec un système Android compatible avec le protocole Android Accessory version 1.

Les applications Android ont quant à elles été testées avec plusieurs téléphones du marché. Les tests de fonctionnement globaux du système sont résumés ci-dessous :

Téléphones	Version	Description	Résultats
Google Nexus S, Google Galaxy Nexus, HTC Desire S	2.3.4, 2.3.6, 4.0.4	L'application est compatible avec tous les téléphones. La communication USB via ADK est fonctionnelle. Le comportement de l'application est identique sur tous les téléphones.	✓
Samsung Galaxy Ace, HTC Desire	2.3.2, 2.3.3	Android 2.3.2 et 2.3.3 ne sont pas compatibles avec le protocole Android Accessory. Les téléphones ne sont donc pas détectés par l'accessoire USB. Aucune communication n'est possible. Ils peuvent seulement être rechargés.	✓

Table 8.1 – Tests de couplage entre Android et l'accessoire USB

Les résultats attendus ont été obtenus. Le couplage Android - accessoire est réussi pour les appareils équipés d'Android 2.3.4 et plus.

Le Samsung Galaxy Ace et le HTC Desire sont des téléphones qui ne disposent malheureusement pas d'une version d'Android assez récente pour supporter le protocole Android Accessory. Ils ne sont donc pas détectés par l'accessoire.

Par exemple, HTC n'a pas prévu de mettre à jour le Desire téléphone vers Android 4 car il est jugé trop vieux. Il est remplacé par des modèles plus récents comme le HTC Desire S, Desire HD, etc. qui eux sont compatibles.

8.2 Problèmes rencontrés

Au départ du projet, des tests ont été réalisés avec le stack USB Host d'exemple fournit par NXP. Des modifications ont dû y être apportées pour qu'il soit plus robuste et qu'il puisse être intégré dans un projet plus complexe. Un effort particulier a été entrepris afin de supprimer les méthodes bloquantes et de mieux gérer les erreurs et les exceptions.

Le processeur utilisé sur la carte *uGates-mini* dispose d'un total de 64 kB de SRAM. Elle est divisée en 3 sections disctintes. 32 kB sont disponibles pour le code et 2x16 kB pour les périphériques. Les différents

buffer utilisés ont dû être répartis dans ces différentes mémoires. Des débordements de mémoire ont été constatés lors de l'utilisation de l'allocation dynamique de mémoire. De ce fait, l'espace mémoire nécessaire est alloué de manière statique. L'ajout d'un système d'exploitation embarqué faciliterait la gestion de la mémoire. De plus, des Thread permettraient de mieux découpler les opérations et de rendre plus prioritaires celles qui sont critiques.

La configuration du contrôleur *GPDMA* (General Purpose DMA) du processeur est complexe à configurer du fait de sa généricité. Comme déjà expliqué, il est utilisé pour copier le buffer audio vers le contrôleur I2S. Lorsqu'un transfert est terminé, la nouvelle copie doit être redémarrée manuellement. Le contrôleur dispose d'un système de *linked-list* qui peut être utilisé afin de copier des zones de mémoires non-contiguës. Il semble que ce système ne permette pas de redémarrer automatiquement la copie des données. Des optimisations devront être effectuées afin d'augmenter la qualité de la sortie audio. Les fréquentes interruptions USB et DMA ont aussi posé des problèmes pour le débuguage du code.

Du côté d'Android, trois applications peuvent communiquer avec la docking station via USB. Le cycle de vie de chaque application a dû être correctement géré pour permettre l'utilisation des différentes applications Android avec le même accessoire. Comme décrit au chapitre 7.7.1-Thread de réception des données, la fermeture de la communication avec l'accessoire USB pose encore des problèmes dans certains cas.

Enfin, la fragmentation des versions Android est un problème récurrent. Différentes librairies additionnelles de compatibilité ont dû être utilisées afin de rendre les applications développées compatibles avec le plus grand nombre de versions.

8.3 Améliorations futures

Dans sa version actuelle, l'accessoire USB *uGates-mini* est opérationnel. Les modifications et améliorations décrites dans le chapitre 5.11-Modifications devront être apportées pour la prochaine version du kit. Sa taille pourra être réduite en supprimant par exemple les points de mesure prévus sur le prototype. Les aspect mécaniques n'ont pas été abordés durant ce projet. L'ajout d'un boîtier pour l'accessoire pourrait être envisagé.

Par manque de temps, le JTAG intégré à l'accessoire n'a pas encore été testé. Une sonde JTAG externe a été utilisée et a permis de programmer rapidement le processeur. Il sera testé durant l'été, avant la fabrication de la nouvelle version de l'accessoire.

La sortie audio de la carte *uGates-mini* a été configurée à 16 kHz. Ce choix a permis de tester le hardware de la carte. Pour l'instant, seule cette configuration est disponible. Des optimisations permettront d'améliorer la qualité audio pour atteindre par exemple celle d'un CD audio, soit 44.1 kHz. Plusieurs configurations pour la sortie audio pourraient alors être proposées.

Les E/S des accessoires USB doivent être configurées dans un fichier de compatibilité XML utilisé par l'application Android. Pour être encore plus générique, la configuration des accessoires pourrait être transmise au téléphone lors de sa connexion. Ainsi, les accessoires pourraient être modifiés sans qu'une mise à jour de l'application soit nécessaire. Le firmware de l'accessoire pourrait aussi être mis à jour directement depuis le téléphone, lorsque de nouvelles versions seraient disponibles.

Finalement, de nouvelles fonctionnalités ont été ajoutées dans la version 2 d'Android Accessory. Il serait intéressant d'analyser les modifications et nouveautés apportées afin de rendre l'accessoire USB compatible avec cette nouvelle version.

Chapitre 9

Conclusion

Le couplage entre l'accessoire USB et le système Android est réussi. L'accessoire USB supporte pleinement le protocole Android Accessory spécialement développé par Google pour une telle application. La carte *uGates-mini* fait office de docking station pour tous les appareils Android équipés d'Android 2.3.4 et supérieur, ce qui représente aujourd'hui plus de 75% des systèmes Android disponibles sur le marché, téléphones et tablettes confondus. Les appareils non compatibles disposent d'une version d'Android publiée il y a plus de deux ans, donc avant la spécification du protocole Android Accessory. Le pourcentage des téléphones non compatibles va encore diminuer. Les anciens téléphones qui ne disposent pas de mises à jour vers Android 4 vont ainsi peu à peu disparaître et être remplacés par des modèles plus récents.

Le système générique de ressources mis en place pour partager des données entre le périphérique Android et la docking station permet d'ajouter facilement d'autres entrées/sorties ou capteurs à l'aide de carte d'extension par exemple. Les périphériques Android sont des systèmes embarqués très complexes qui disposent d'une grande puissance de calcul ainsi que d'une grande variété de capteurs et d'interfaces. L'exemple de ce couplage réussi démontre que les systèmes Android peuvent maintenant être intégrés dans des installations existantes pour y ajouter, par exemple, une interface homme-machine ou pour utiliser leurs interfaces et capteurs, tels que caméra, GSM ou Wi-Fi. De plus, le système de compatibilité mis en place permet d'utiliser une application Android avec plusieurs stations d'accueil qui ne disposeraient pas forcément du même hardware.

Le prototype « proof of concept » développé nécessite encore quelques améliorations mais il dispose déjà des fonctions que l'on retrouve sur les docking station vendues sur le marché. La miniaturisation et l'aspect mécanique de la docking station restent à étudier afin d'envisager un produit final commercialisable, pouvant concurrencer les produits disponibles des autres marques de téléphone.

Le bilan du projet est très positif. La fabrication d'une deuxième version de l'accessoire *uGates-mini* est d'ores et déjà prévue dans un prochain projet. Simultanément, plusieurs améliorations pour la deuxième version du protocole Android Accessory ont été présentées par Google et seront disponibles dans la nouvelle version d'Android 4.1 *Jelly Bean*. Google souhaite faciliter le développement de docking station Android, peu de modèles étant disponibles sur le marché. Par exemple, le mode audio développé dans ce projet est maintenant fourni directement par le système d'exploitation grâce à Android Accessory version 2.

Au final, ce projet a permis de démontrer un couplage USB réussi entre la docking station et le système Android et son adéquation avec les développements réalisés par Google.

« Android Open Accessories : A Bright Idea ! »
Source : http ://android-developers.blogspot.com

Sion, le 9 juillet 2012

Christopher Métrailler

Bibliographie

[1] Projet Arduino.
http ://arduino.cc/.

[2] CMSIS Cortex Microcontroller Software Interface Standard ARM.
http ://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php.

[3] Stations d'accueil Philips pour Android.
http ://www.philips.ch/c/peripherique-audio/.

[4] Projet C de démonstration "USB Host Lite" pour LPC1768.
http ://support.code-red-tech.com/CodeRedWiki/RDB1768ExampleProjects.

[5] Etude de marché de comparis.ch sur les smartphones en suisse (12 mars 2012).
http ://www.comparis.ch/ /media/files/mediencorner/medienmitteilungen/2012/telecom/verbreitung-smartphone.pdf.

[6] Installation du SDK Android.
http ://developer.android.com/sdk/index.html.

[7] Versions et part de marché Android.
http ://developer.android.com/resources/dashboard/platform-versions.html.

[8] libmad on Android with the NDK.
http ://www.badlogicgames.com/wordpress/ ?p=231.

[9] Cortex M3 NXP LPC1768.
http ://www.nxp.com/products/microcontrollers/cortex-m3/lpc1700/LPC1768FBD100.html.

[10] Projet mbed. Rapid Prototyping for Microcontrollers.
http ://mbed.org/.

[11] Norme USB 2.0 non officielle simplifiée.
http ://www.beyondlogic.org/usbnutshell/.

[12] Documentation officielle d'Android Accessory.
http ://developer.android.com/guide/topics/usb/index.html.

[13] Annonce officielle de Android Open Accessories sur le blog Android.
http ://android-developers.blogspot.com/2011/05/bright-idea-android-open-accessories.html.

[14] Spécification officielle USB.
http ://www.usb.org/developers/docs/.

[15] Projet IOIO open source pour Android.
https ://github.com/ytai/ioio/wiki.

[16] Wave PCM soundfile format.
https ://ccrma.stanford.edu/courses/422/projects/WaveFormat/.

[17] Open Host Controller Interface Specification.
http ://h18000.www1.hp.com/productinfo/development/openhci.html.

[18] Android Wikipedia.
http ://fr.wikipedia.org/wiki/Android.

[19] Documentation USB Wikipedia.
http ://fr.wikipedia.org/wiki/USB.

Chapitre 10

CD-ROM

Logiciels */software/*

TheDotFactory - An LCD Font and Image Generator
Audacity - Software for recording and editing sounds
FT Prog - EEPROM Programming Utility for FTDI devices
USBLyzer - USB Protocol Analyzer and USB Traffic Sniffer

Code source */src/*

Workspace Android
Accessoire USB *uGates-mini*

Environnement de développement */toolchain/*

Eclipse C++
OpenOCD
LPC17xx CMSIS driver library
Sourcery CodeBench Lite

PCB */pcb/*

PCB uGates HEVs
Accessoire *uGates-mini*
Carte d'extension *MezzaLCD*

Datasheet */datasheet/*

Stereo Headphone Driver - *MAX4410*
Liquid Crystal Display Module - *NHD-C12832A1Z*
Low power audio DAC - *UDA1334BT*
LPC17xx User manual - *UM10360*

Chapitre 11

Annexes

A Parts de marché et historique des versions Android du 2 avril 2012.
B Proposition de travail de diplôme.
C Donnée du travail de diplôme.
D Setting up eclipse for ARM-programming on Windows.
E Dossier de fabrication de l'accessoire USB *uGates-mini*.
F Dossier de fabrication de la carte d'extension.
G Diagramme de classes - accessoire USB.
H Code de la classe *Resource*.
I Spécification du protocole de l'application.
J Thread de réception des données sur Android.
K Thread audio sur Android.

Annexe A

Parts de marché et historique des versions Android du 2 avril 2012.

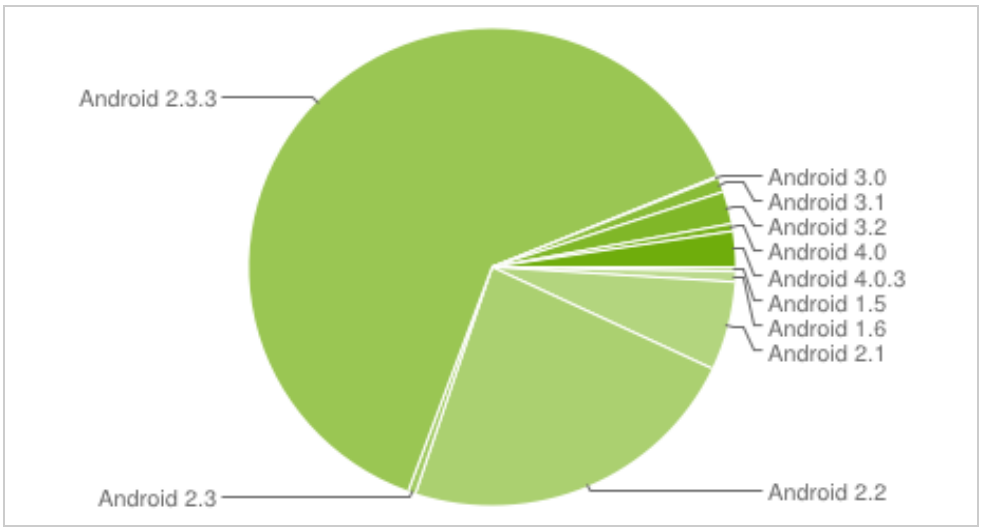
android developers

Platform Versions

This page provides data about the relative number of active devices running a given version of the Android platform. This can help you understand the landscape of device distribution and decide how to prioritize the development of your application features for the devices currently in the hands of users. For information about how to target your application to devices based on platform version, see API Levels.

Current Distribution

The following pie chart and table is based on the number of Android devices that have accessed Google Play within a 14-day period ending on the data collection date noted below.



Platform	Codename	API Level	Distribution
Android 1.5	Cupcake	3	0.3%
Android 1.6	Donut	4	0.7%
Android 2.1	Eclair	7	6.0%
Android 2.2	Froyo	8	23.1%
Android 2.3 - Android 2.3.2	Gingerbread	9	0.5%
Android 2.3.3 - Android 2.3.7		10	63.2%
Android 3.0	Honeycomb	11	0.1%
Android 3.1		12	1.0%
Android 3.2		13	2.2%
Android 4.0 - Android 4.0.2	Ice Cream Sandwich	14	0.5%
Android 4.0.3		15	2.4%

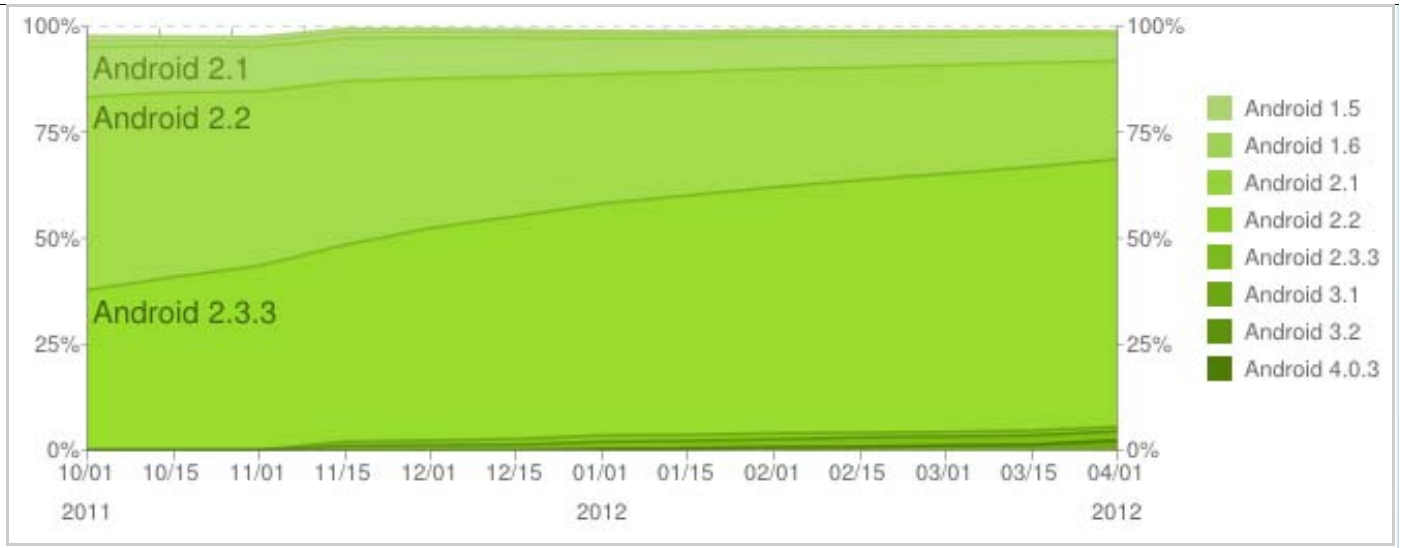
Data collected during a 14-day period ending on April 2, 2012

Historical Distribution

The following stacked line graph provides a history of the relative number of active Android devices running different versions of the Android platform. It also provides a valuable perspective of how many devices your application is compatible with, based on the platform version.

Notice that the platform versions are stacked on top of each other with the oldest active version at the top. This format indicates the total percent of active devices that are compatible with a given version of Android. For example, if you develop your application for the version that is at the very top of the chart, then your application is compatible with 100% of active devices (and all future versions), because all Android APIs are forward compatible. Or, if you develop your application for a version lower on the chart, then it is currently compatible with the percentage of devices indicated on the y-axis, where the line for that version meets the y-axis on the right.

Each dataset in the timeline is based on the number of Android devices that accessed Google Play within a 14-day period ending on the date indicated on the x-axis.



Last historical dataset collected during a 14-day period ending on April 2, 2012

↑ Go to top

Medienmitteilung

comparis.ch zur Verbreitung von Smartphones

2,9 Millionen Schweizer haben ein Smartphone

Rund 48 Prozent oder praktisch jeder zweite Schweizer besitzt ein iPhone oder ein anderes Smartphone. Bei den jungen Erwachsenen sind es sogar rund vier von fünf Personen. Das geht aus einer repräsentativen Umfrage des Internet-Vergleichsdienstes comparis.ch unter mehr als 1200 Personen hervor.

Zürich, 20. März 2012 – Der eine hat's, der andere nicht; im Alltag präsent ist das Smartphone alleweil. Doch wie stark verbreitet sind iPhones, Android-Geräte und weitere multimediale Tausendsassas tatsächlich? comparis.ch wollte es wissen und hat dazu in einer repräsentativen Umfrage mehr als 1200 Personen zwischen 15 und 74 Jahren befragt. Durchgeführt wurde die Umfrage durch das Marktforschungsinstitut Link Ende Februar 2012. Aus den Angaben geht hervor, dass heute praktisch jeder zweite Schweizer ein Smartphone besitzt: 48 Prozent der Befragten gaben an, ein Mobiltelefon mit Touchscreen und Internetzugang zu haben. Hochgerechnet auf die ständige Schweizer Wohnbevölkerung zwischen 15 und 74 Jahren, haben also 2,9 Millionen Personen ein Smartphone. Noch Ende 2007 besassen lediglich 3 Prozent der Befragten ein Smartphone. Dies zeigen die Antworten auf die Frage, wann sich die Smartphone-Besitzer erstmals ein solches Gerät gekauft haben.

Rasantes Wachstum innert nur vier Jahren

«Die vielseitigen Funktionen des Smartphones überzeugen offenbar immer mehr Leute», sagt Ralf Beyeler, Telecom-Experte von comparis.ch, als Erklärung für die enorme Verbreitung. Mit dem Eintritt von Apple in den Schweizer Markt Mitte 2008 setzte die Ausbreitung in vollen Zügen ein. Seither steigt die Kurve steil an (Grafik 1): 23 Prozent der Smartphone-Besitzer haben ihr erstes Smartphone im Jahr 2010 gekauft, 37 Prozent im Jahr 2011. Mit anderen Worten: 60 Prozent und damit die weitaus meisten Smartphone-Nutzer haben ihr erstes Gerät erst in diesen zwei Jahren angeschafft.

«Smartphones gab es schon früher, doch Apple hat den Geräten innerhalb von nur vier Jahren zum Durchbruch verholfen», sagt Ralf Beyeler. Aber auch die Geräte mit Android-Betriebssystem haben wesentlich zur Verbreitung der Smartphones beigetragen. Der Wettbewerb zwischen den Anbietern dürfte ein Grund für die schnelle Verankerung der Geräte im Alltag sein. Allerdings ist zu erwarten, dass sich das Wachstum in Zukunft nicht mehr so rasant gestaltet. «Wahrscheinlich werden sich 2012 weniger Menschen als in den letzten Jahren erstmals ein Smartphone kaufen», meint Ralf Beyeler. Denn nicht zu vergessen ist, dass viele Leute gar nicht das Bedürfnis nach ständigem Internetzugang und unzähligen Apps haben; es gibt nach wie vor Leute, die bloss telefonisch erreichbar sein möchten – zum Beispiel in einem Notfall – und darum mit herkömmlichen Handys zufrieden sind.

Vier von fünf Jugendlichen haben ein Smartphone

Schliesslich könnte bei der wichtigen Zielgruppe der jungen Erwachsenen der Markt bald gesättigt sein. Mittlerweile besitzen fast vier von fünf Jugendlichen zwischen 15 und 19 Jahren ein Smartphone (Grafik 2), bei den Personen zwischen 20 und 24 Jahren sind es etwa drei von vier. Unter 50 Prozent fällt der Anteil Smartphone-Besitzer erst bei den 40- bis 44-Jährigen. «Junge Leute sind in einer digitalen Welt aufgewachsen und möchten auch unterwegs über das Internet mit ihren Freunden verbunden sein», sagt Ralf Beyeler. Von den Personen über 70 Jahren nutzen aber immer noch 11 Prozent ein Smartphone. Überraschend ist, dass im Tessin das Smartphone weniger stark verbreitet ist als in der übrigen Schweiz. Nur rund 36 Prozent der Tessiner haben

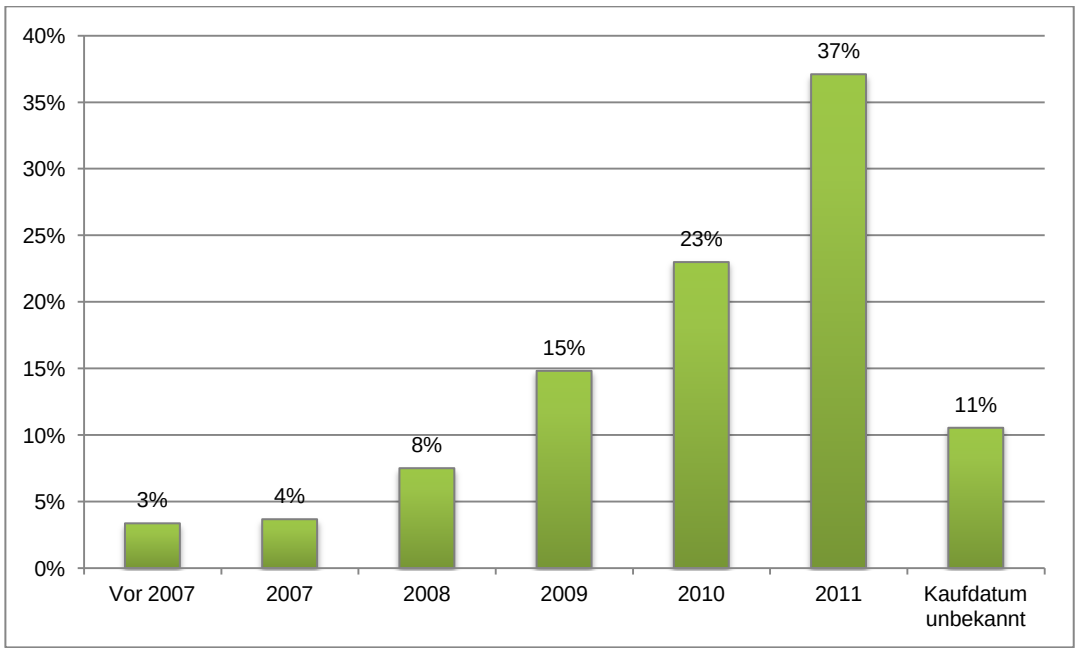
comparis.ch AG
Stampfenbachstrasse 48
CH-8006 Zürich
T: +41 44 360 34 00
F: +41 44 360 52 72
media@comparis.ch

eines, gegenüber 50 Prozent der Romands und 48 Prozent der Deutschschweizer. Weniger überraschend dagegen: Mehr Männer (53 Prozent) als Frauen (42 Prozent) haben ein Smartphone.

Apple dominiert den Markt
Das mit Abstand am meisten verwendete Smartphone ist das iPhone (Grafik 3). 55 Prozent der Smartphone-Besitzer verwenden das Apple-Produkt. Die grösste Konkurrentin kommt auf einen Anteil von etwas mehr als einem Drittel: 36 Prozent haben ein Gerät mit Android-Betriebssystem. Bedeutungslos ist demgegenüber die Verbreitung der Geräte mit anderen Betriebssystemen.

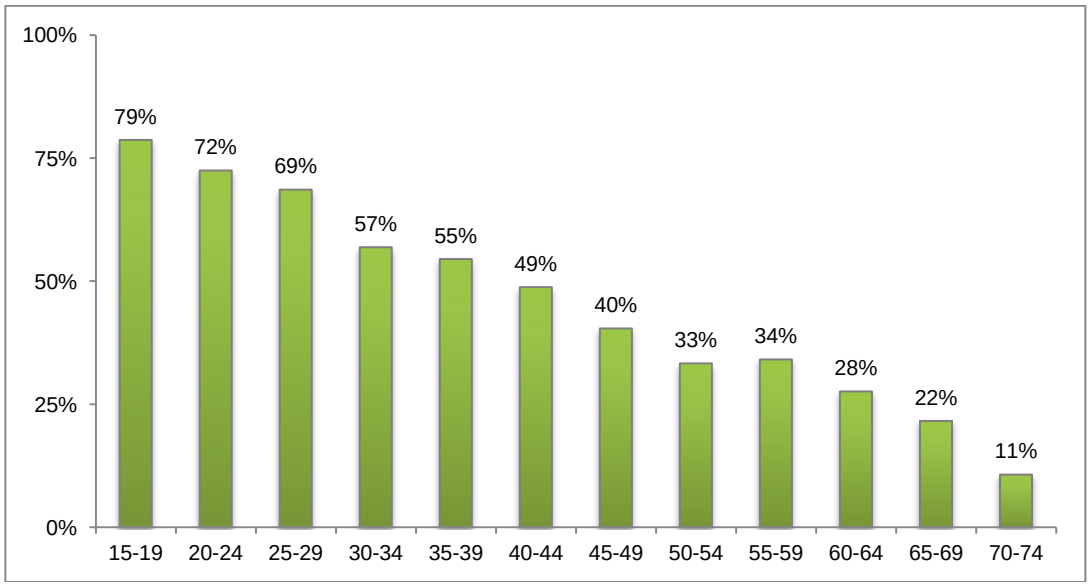
Weitere Informationen:
Ralf Beyeler
Telecom-Experte
Telefon: 044 360 52 77
Handy: 079 467 07 81
E-Mail: ralf.beyeler@comparis.ch
www.comparis.ch

Grafik 1: Zeitpunkt des ersten Smartphone-Kaufs (bis 2011)
(Anteil an allen Smartphone-Besitzern)



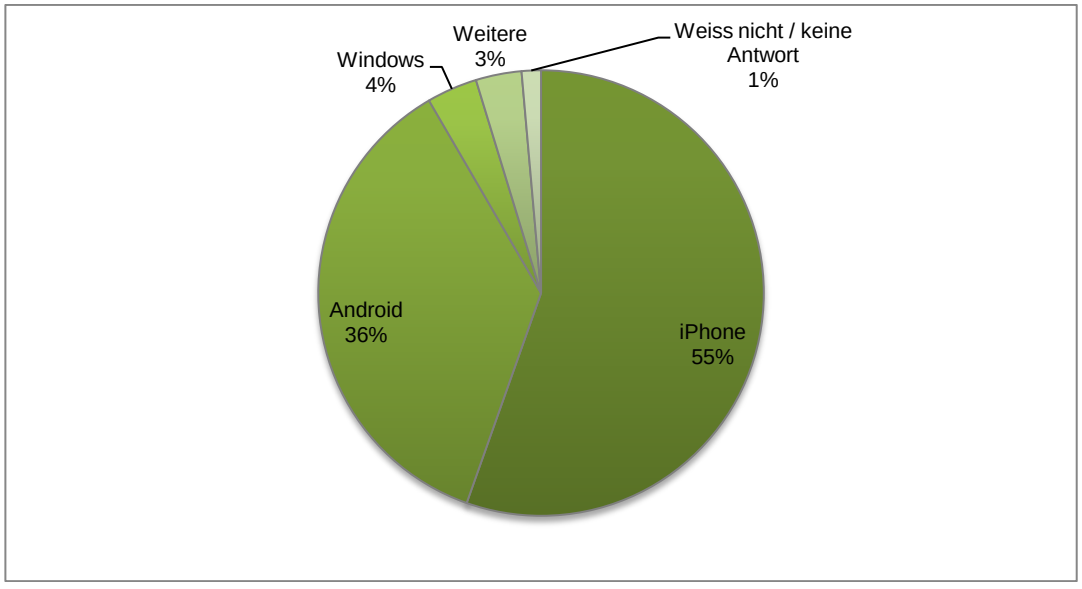
n=540 Quelle: comparis.ch

Grafik 2: Anteil Smartphone-Besitzer nach Altersgruppen



N=1202 Quelle: comparis.ch

Grafik 3: Anteil der Smartphone-Betriebssysteme



n=572 Quelle: comparis.ch

Annexe B

Proposition de travail de diplôme.

Android Docking Station

Mots-clés : Android, JAVA, USB, ADK, ADB, programmation C/C++, uP

De plus en plus de téléphones Android sont présents sur le marché. En parallèle de nouveaux accessoires voient le jour. Ces téléphones sont tous équipés d'un port USB qui permet de les configurer et de les utiliser via un PC pour y transférer des données, par exemple.

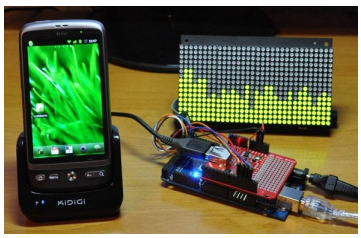


FIGURE 1 – Analyse spectrale d'un signal audio sur une matrice à LEDs 32x16.

L'idée de ce projet est de concevoir un système embarqué communiquant en USB avec le téléphone. Cette docking station sera compatible avec la majeure partie des téléphones Android et pourra remplacer un PC pour différentes applications. Via une application Android, des données pourront être transmises ou reçues sur le téléphone. Un grand nombre d'applications sont envisageables, dont voici quelques exemples :

- Docking station pour téléphones Android (exemples : centrale météo, lecteur audio/réveil)
- Commande et contrôle de périphériques via Android
- Cartes d'extension pour téléphones (entrées/sorties, lecteur NFC, adaptateur réseau)

Tâches à réaliser

- Communication USB entre le téléphone et le uP (programmation C/C++)
- Interfaçage d'un contrôleur USB Host ou module Hardware
- Communication avec Android Open Accessories (ADK), compatible Android 2.3.4 et plus récent
- Communication avec Android Debug Bridge (ADB), compatible Android 1.5 et plus récent
- Développement d'une application Android (programmation JAVA, Android SDK)

Références

[1] Android Debug Bridge (ADB) : http ://developer.android.com/guide/developing/tools/adb.html

[2] Android Open Accessory Development Kit (ADK) : http ://accessories.android.com/

[3] ADB implementation for microcontrollers : http ://code.google.com/p/microbridge/

[4] IOIO for Android : https ://github.com/ytai/ioio/wiki

[5] Seeeduino ADK Main Board : http ://www.seeedstudio.com/depot/seeeduino-adk-main-board-p-846.html ?cPath=132_133

[6] USB Host Shield : http ://www.circuitsathome.com/arduino_usb_host_shield_projects

[7] Illustration : http ://adk-open-call-2011.blogspot.com/2011/09/blog-post.html

Annexe C

Donnée du travail de diplôme.

Données du travail de diplôme
Daten der Diplomarbeit

☒ FSI ☐ FTV	Année académique / *Studienjahr* **2011/2012**	No TD / *Nr. DA* **it/2012/23**
Mandant / *Auftraggeber* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*	Etudiant / *Student* **Christopher Métrailler** Professeur / *Dozent* **Pierre-André Mudry**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein	Expert / *Experte (données complètes)* **Prof. Fabien Vannel** Hepia | Rue de la Prairie 4 | 1202 Genève	

Titre / *Titel*

Android docking station

Description et Objectifs / *Beschreibung und Ziele*

De plus en plus de téléphones Android sont présents sur le marché. En parallèle, de nouveaux accessoires voient le jour. Ces téléphones sont tous équipés d'un port USB qui permet de les configurer et de les utiliser via un PC pour y transférer des données.

L'objectif du projet est de concevoir un système embarqué communicant en USB avec le téléphone et permettant d'être utilisé comme « docking station ». Cette station sera compatible avec la majeure partie des téléphones Android et pourra être utilisée pour différentes applications comme une centrale météo, un lecteur audio amplifié, un réveil, etc. Le système embarqué réalisé devra être suffisamment modulaire pour pouvoir supporter l'ajout de différents types de capteur (température, hygrométrie, GPIO, autre...) pouvant être accédés depuis Android.

Les objectifs sont :

— Test de la carte électronique modulaire développée durant le projet de semestre
— Développement du logiciel générique Android d'intégration des périphériques
— Permettre l'accès à quelques périphériques Android du côté de la docking station
— Démonstrateur « proof of concept » permettant de démontrer le couplage entre la docking station et le système Android.

Délais / *Termine*

Attribution du thème / *Ausgabe des Auftrags:*
14.05.2012

Exposition publique / *Ausstellung Diplomarbeiten:*
31.08.2012

Remise du rapport / *Abgabe des Schlussberichts:*
09.07.2012 | 12h00

Défense orale / *Mündliche Verteidigung:*
Semaine / *Woche 36*

Signature ou visa / *Unterschrift oder Visum*

Responsable de l'orientation
Leiter der Vertiefungsrichtung: .....

[1] Etudiant/*Student*: Métrailler Ch.

[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive et le caractère confidentiel du travail de diplôme qui lui est confié et des informations mises à sa disposition.*
Durch seine Unterschrift verpflichtet sich der Student, die Richtlinie einzuhalten sowie die Vertraulichkeit der Diplomarbeit und der dafür zur Verfügung gestellten Informationen zu wahren.

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats* .....

Annexe D

Setting up eclipse for ARM-programming on Windows.

Setting up eclipse for ARM-programming on Windows - V2

Update of the tutorial "Setting up eclipse for ARM-programming on Windows".

1	**First revision.**	**fud**	08.05.2011
2	**Second revision.** - Software updates. - Tested on Windows 7	**Christopher Métrailler**	24.02.2012

1. Installing the environnement

PC configuration: Windows 7 Professionnel SP1 32bit.
Target: uGates developpement board - LPC1768 and Amontec USB JTAGkey.

- **Sourcery G++ Lite 2010.09-51**

  Install code sourcery version - *D:\ARM\arm-2010.09*
  Add bin folder to Windows PATH.
  http://www.codesourcery.com/sgpp/lite/arm/portal/subscription3053

- **Installation of OpenOCD 0.4.0**

  D:\ARM\openocd-0.4.0
  Add bin folder to Windows PATH.
  http://www.freddiechopin.info/index.php/en/download/category/4-openocd

- **JTAG drivers (Amontec JTAGkey)**

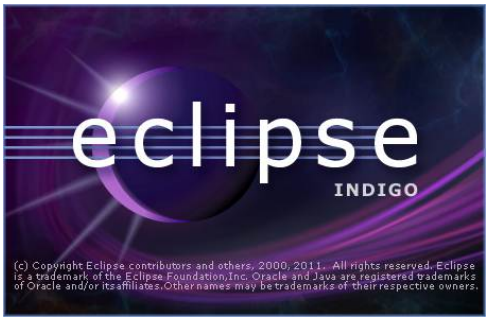
  Install OpenOCD libusb drivers and Amontec drivers.
  D:\ARM\openocd-0.4.0\drivers\libusb-win32_ft2232_driver-100223

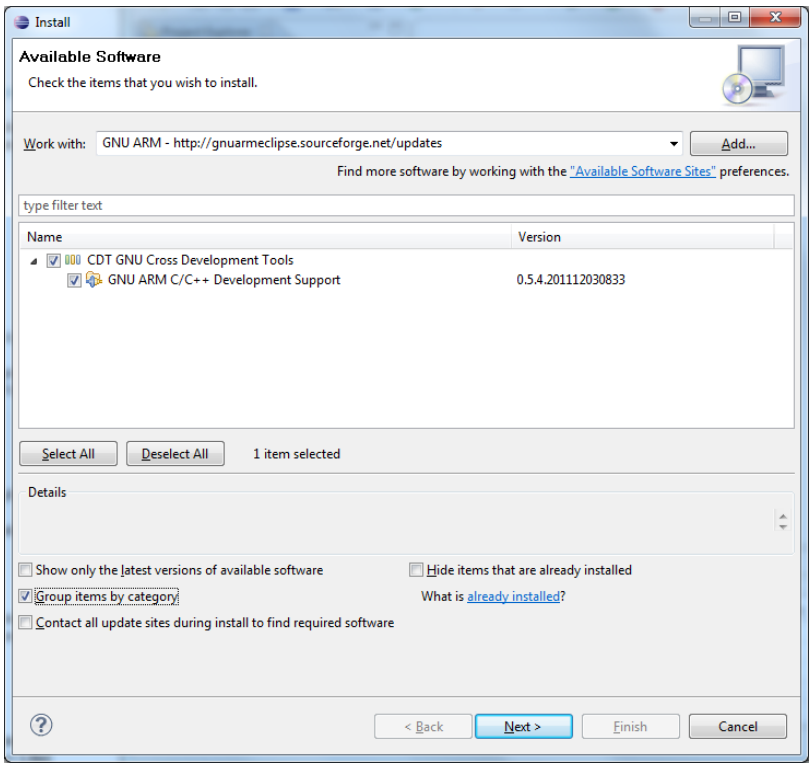
- **Eclipse IDE for C/C++ Developers - Eclipse Helios 32bits**

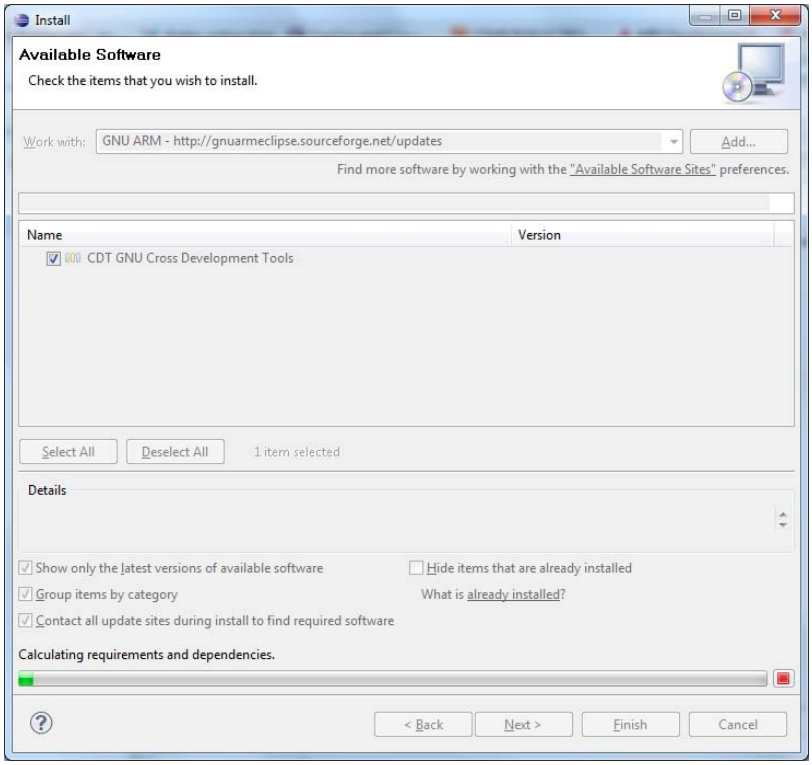
  Download and install the C++ version with CDT (not JAVA) - *D:\ARM\eclipse*
  Set Workspace to *D:\ARM\projects*
  http://www.eclipse.org/downloads

- **GNU ARM - Eclipse Plugin**

  – Open eclipse.
  – Go to Help / Install New Software.
  – Select the name site (http://gnuarmeclipse.sourceforge.net/updates) into “Work with” for install gnu cross development tools for arm.

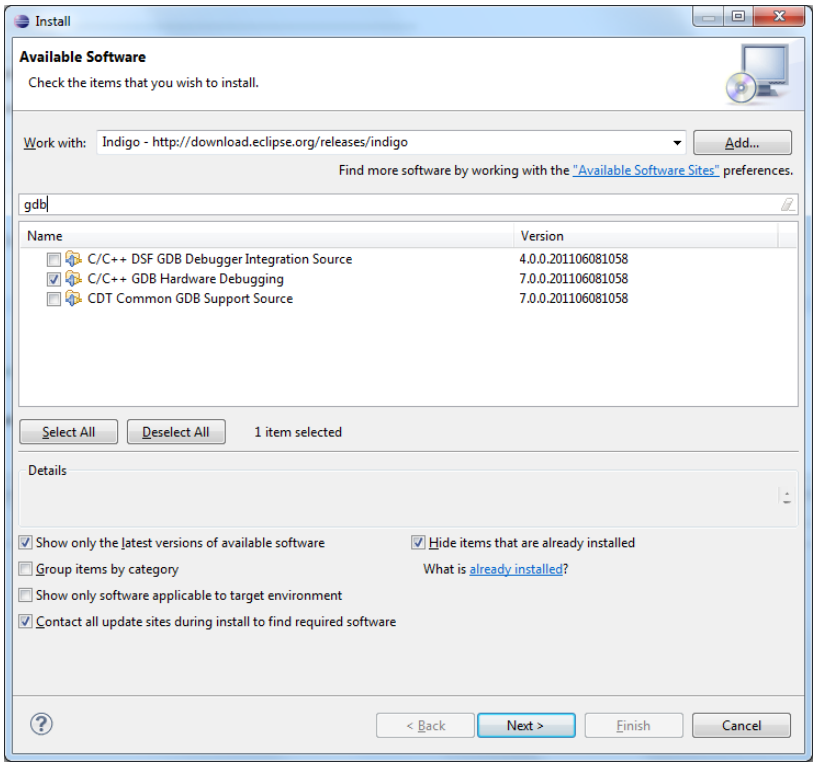


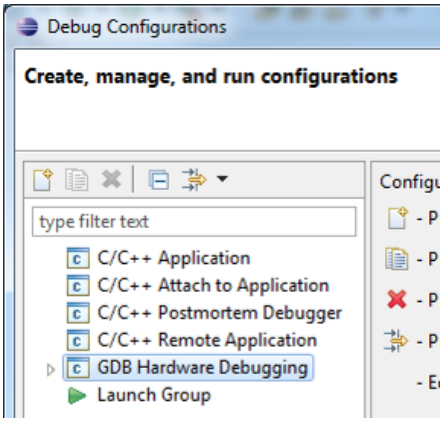

Install
Available Software
Check the items that you wish to install.
Work with: GNU ARM - http://gnuarmeclipse.sourceforge.net/updates
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
Version
CDT GNU Cross Development Tools
GNU ARM C/C++ Development Support
0.5.4.201112030833
Select All
Deselect All
1 item selected
Details
Show only the latest versions of available software
Hide items that are already installed
Group items by category
What is already installed?
Contact all update sites during install to find required software
< Back
Next >
Finish
Cancel


Install
Available Software
Check the items that you wish to install.
Work with: GNU ARM - http://gnuarmeclipse.sourceforge.net/updates
Add...
Find more software by working with the "Available Software Sites" preferences.
Name
Version
CDT GNU Cross Development Tools
Select All
Deselect All
1 item selected
Details
Show only the latest versions of available software
Hide items that are already installed
Group items by category
What is already installed?
Contact all update sites during install to find required software
Calculating requirements and dependencies.
< Back
Next >
Finish
Cancel

- **GDB Hardware Debugging - Eclipse Plugin**

  – Install plugin. Go to Help / Install New Software.



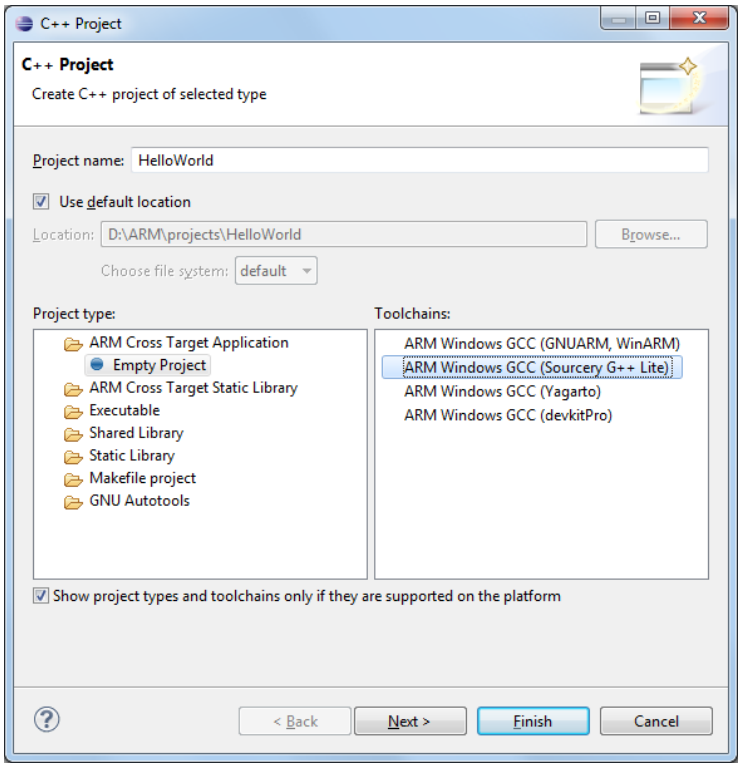


- **LPC17xx CMSIS**

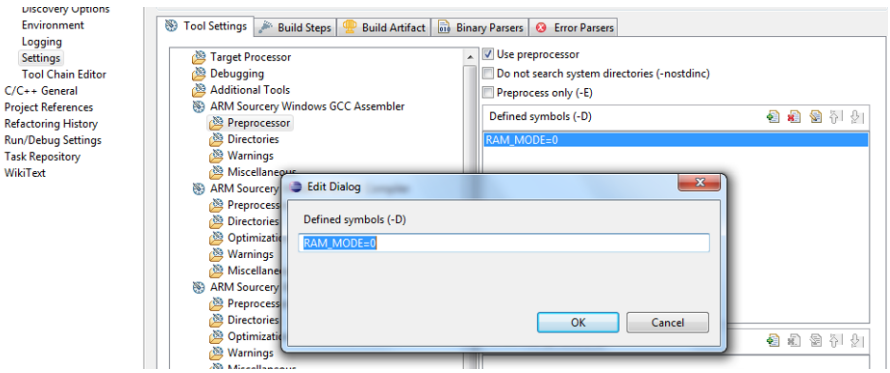
  Download and install - *D:\ARM\CMSIS*
  http://support.code-red-tech.com/CodeRedWiki/NXPDriverLibraries

2. How to set up a project on Eclipse

- **New C++ project**
  - Create the New Project in Eclipse.
  - New Project.
  - C++-Project.
  - Enter project title and choose “ARM-Windows GCC (Sourcery G++ Lite)”.
  - Finish.

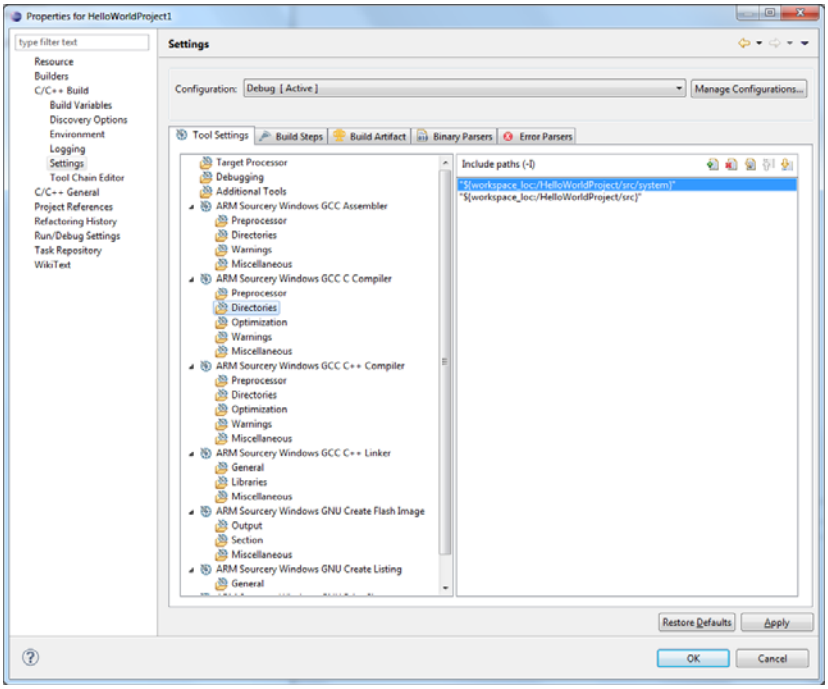


- **Change the project settings**
  - Open Project / Properties.
  - Open the C/C++ Build part.
  - Go to Settings - Tool Settings / Debugging set Debug format to gdb.
  - Tool Settings / ARM Sourcery Windows GCC Assembler / Preprocessor add Defined symbol “RAM_MODE=0”.

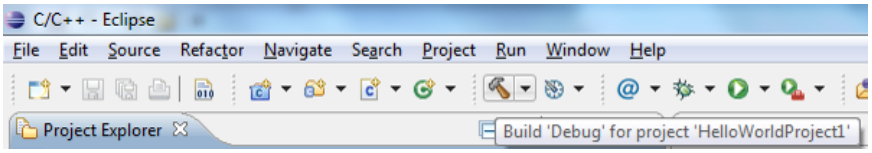


  - Tool Settings / ARM Sourcery Windows GCC C++ Linker / General add: *D:\ARM\CMSIS\makesection\makerule\example\ldscript_rom_gnu.ld*

- **Add needed libraries and include files**
  - Copy folder include and source from an existing project
    D:\ARM\CMSIS\Drivers
  - Copy *lpc17xx_libcfg.h* and *startup_LPC17xx.S* from an existing project:
    D:\ARM\CMSIS\Core\CM3\DeviceSupport\NXP\LPC17xx\startup\startup_LPC17xx.S
    D:\ ARM\CMSIS\Exemples\Project_Templates\Eclipse_GNU\lpc17xx_libcfg.h
  - Add the folders include and source under Properties / C/C++ Build / Tool Settings / Arm Sourcery Windows GCC C Compiler



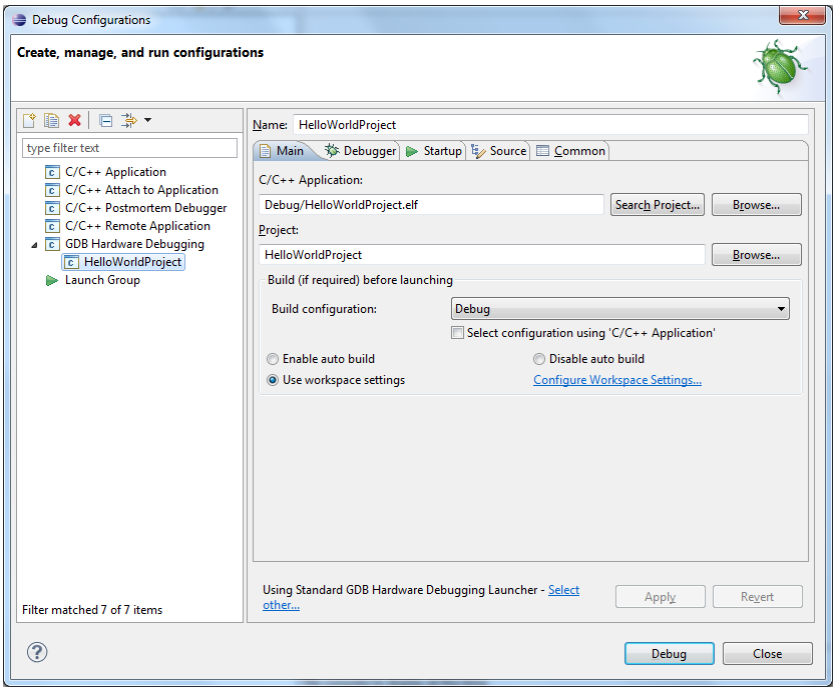
- **Build the project**
  - Copy an existing source file from an existing project and build it.

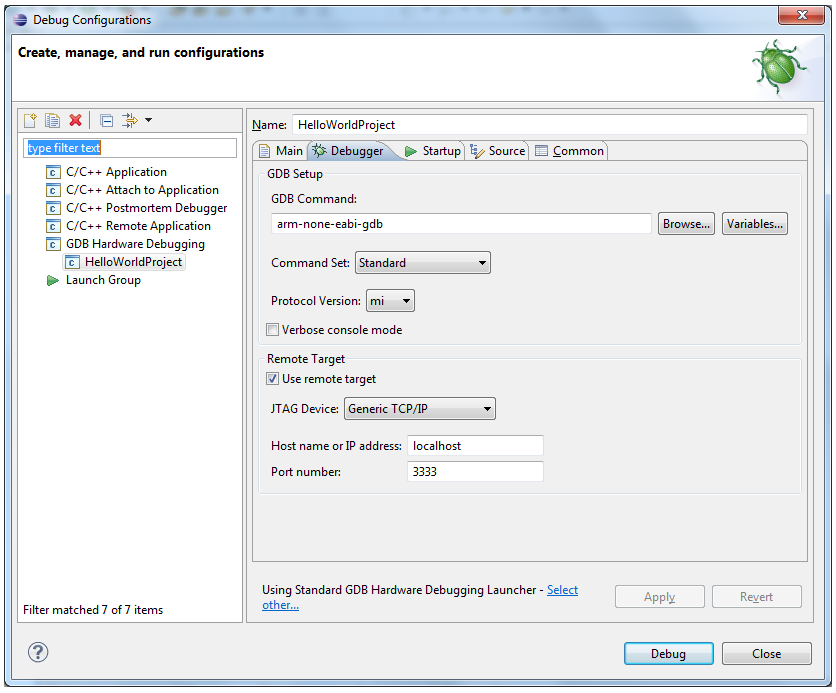


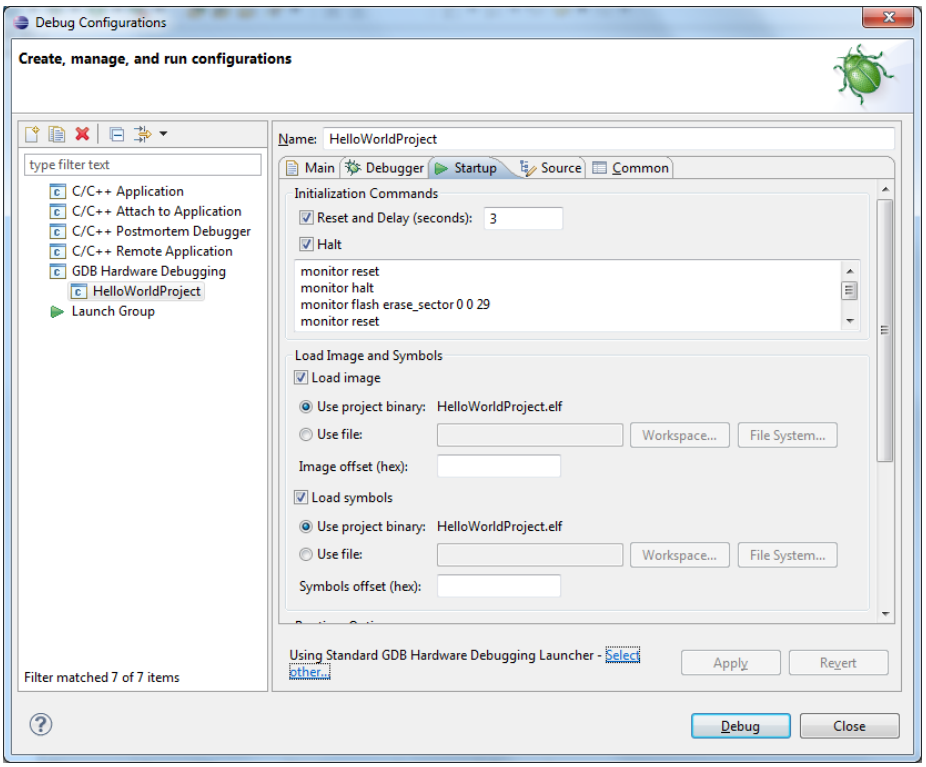
3. Set up arm-none-eabi-gdb for debugging

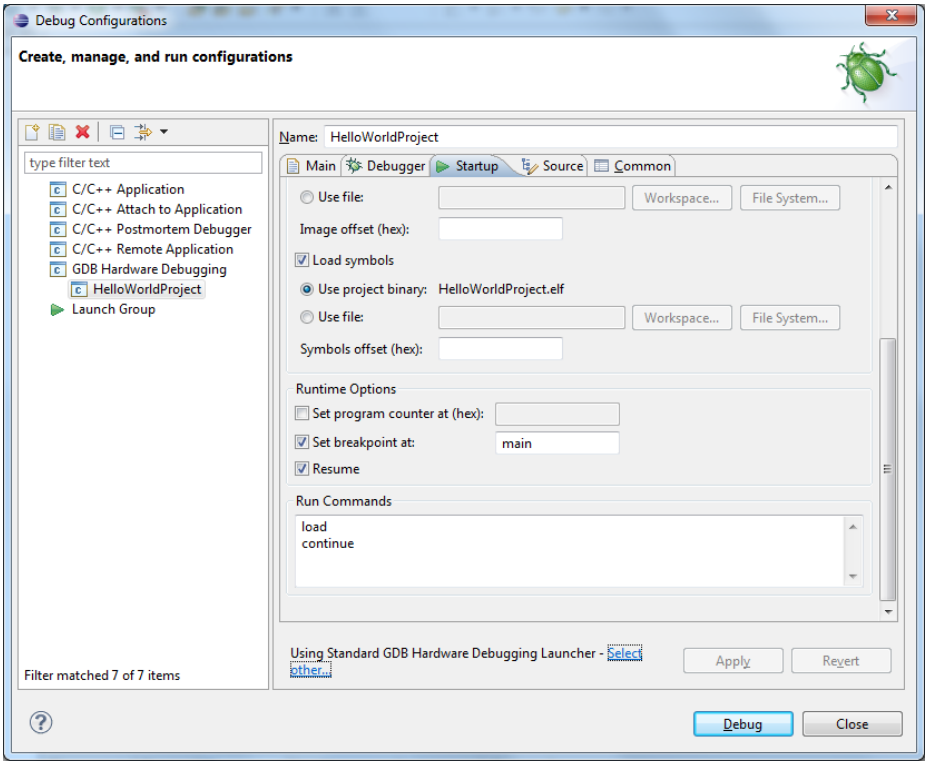
The following screenshots show the setup for the arm gdb debugging with the example HelloWorldProject.elf.

- **Edit the debug configuration**
  - Under Run / Debug Configurations select the new project by clicking on browse beside Project:-Box.
  - Select the right .elf-file by clicking on Search Project right to the C/C++ Application:-Box.
  - Select “Standard GDB Hardware Debugging Launcher”.









- Initialization Commands :

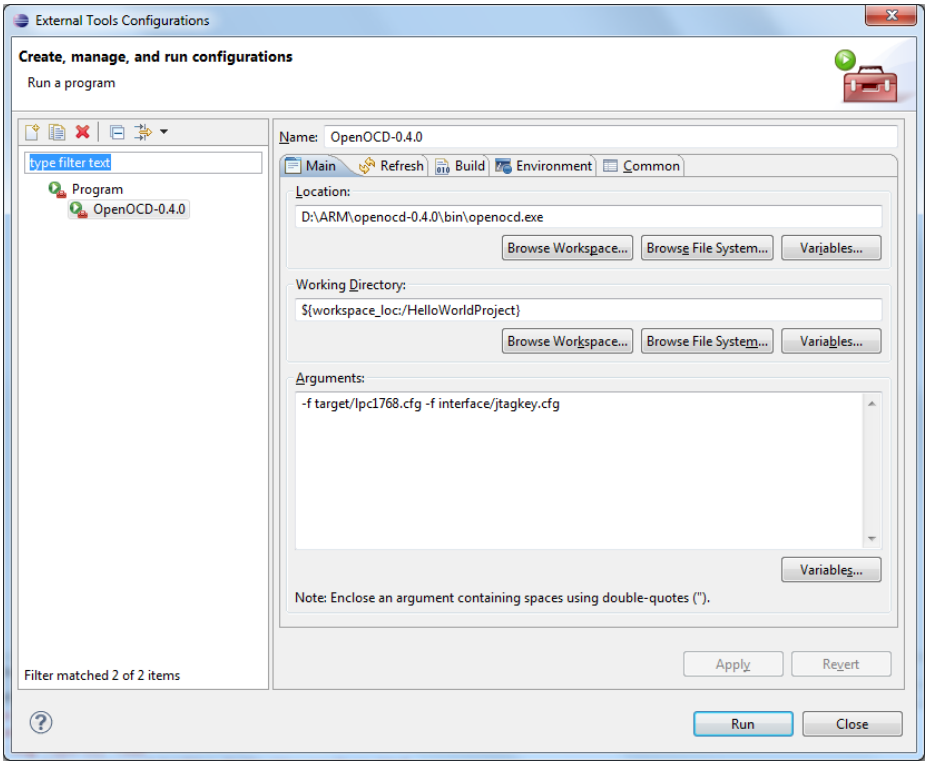
```
monitor reset
monitor halt
monitor flash erase_sector 0 0 29
monitor reset
monitor halt
```

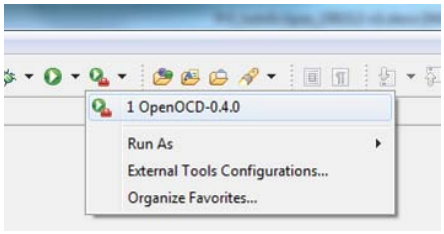
- Run Commands :

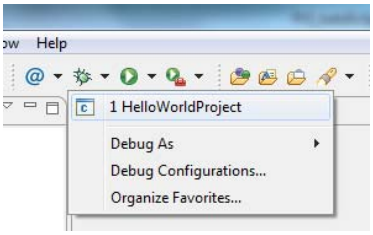
```
load
continue
```

4. Set up Openocd 0.4.0

- Open the External Tools Configurations under Run / externalTools.
- Add a new and name it OpenOCD 0.4.0. Add it to favorite.
- Enter location: *D:\ARM\openocd-0.4.0\0.4.0\bin\openocd.exe*
- Enter the launch arguments:
  -f target/lpc1768.cfg -f interface/jtagkey.cfg







Launch “OpenOCD-0.4.0”. Console output is :

```
Open On-Chip Debugger 0.4.0 (2010-02-22-19:05)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.berlios.de/doc/doxygen/bugs.html
jtag_nsrst_delay: 200
jtag_ntrst_delay: 200
trst_and_srst srst_pulls_trst srst_gates_jtag trst_push_pull srst_open_drain
500 kHz
Info : clock speed 500 kHz
Info : JTAG tap: lpc1768.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0x4)
Info : lpc1768.cpu: hardware has 6 breakpoints, 4 watchpoints
```

Annexe E

Dossier de fabrication de l'accessoire USB *uGates-mini*.

uGates mini v0.1

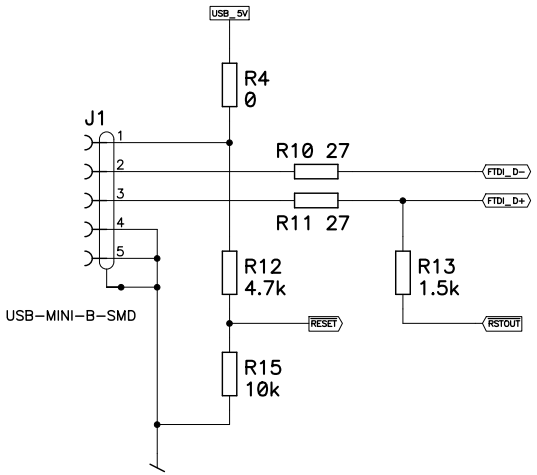
Android Accessory

Page	Content
1	This page
2	JTAG
3	USB Host
4	CPU
5	5V – 3.3V Power supply
6	IO, LEDs, Buttons, Audio

uGates – mini Android Accessory Sheet1	DES	04.04.2012 metraic3
	REV	v0.1
HAUTE ECOLE VALAISANNE	1/6	P:\PCB\Student\TSTD\ uGates-miniV0_1_120412.sch

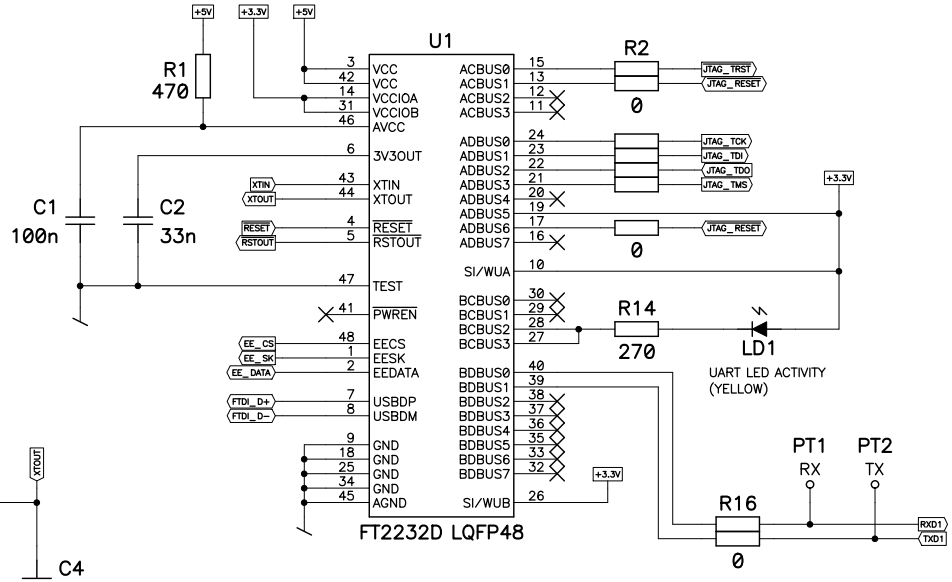
USB Device – JTAG & UART

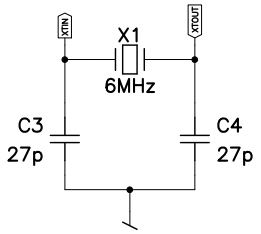
USB device connector



Interfacing to 3.3V Logic

Self Powered Circuit with 3.3V logic drive and IO supply voltage.





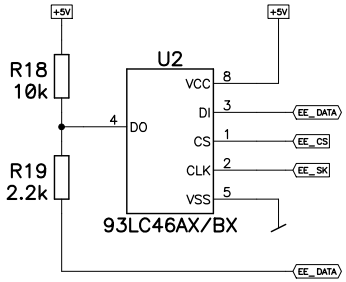
Channel A: JTAG

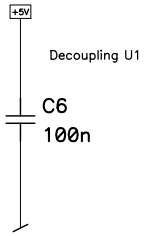
- $\overline{\text{TRST}}$
- $\overline{\text{RESET}}$
- TCK
- TDI
- TDO
- TMS

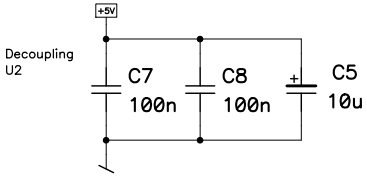
Channel B: UART

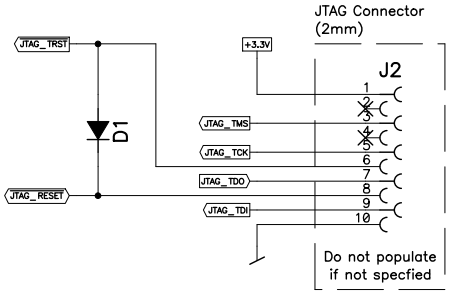
- RX
- TX

EEPROM Configuration



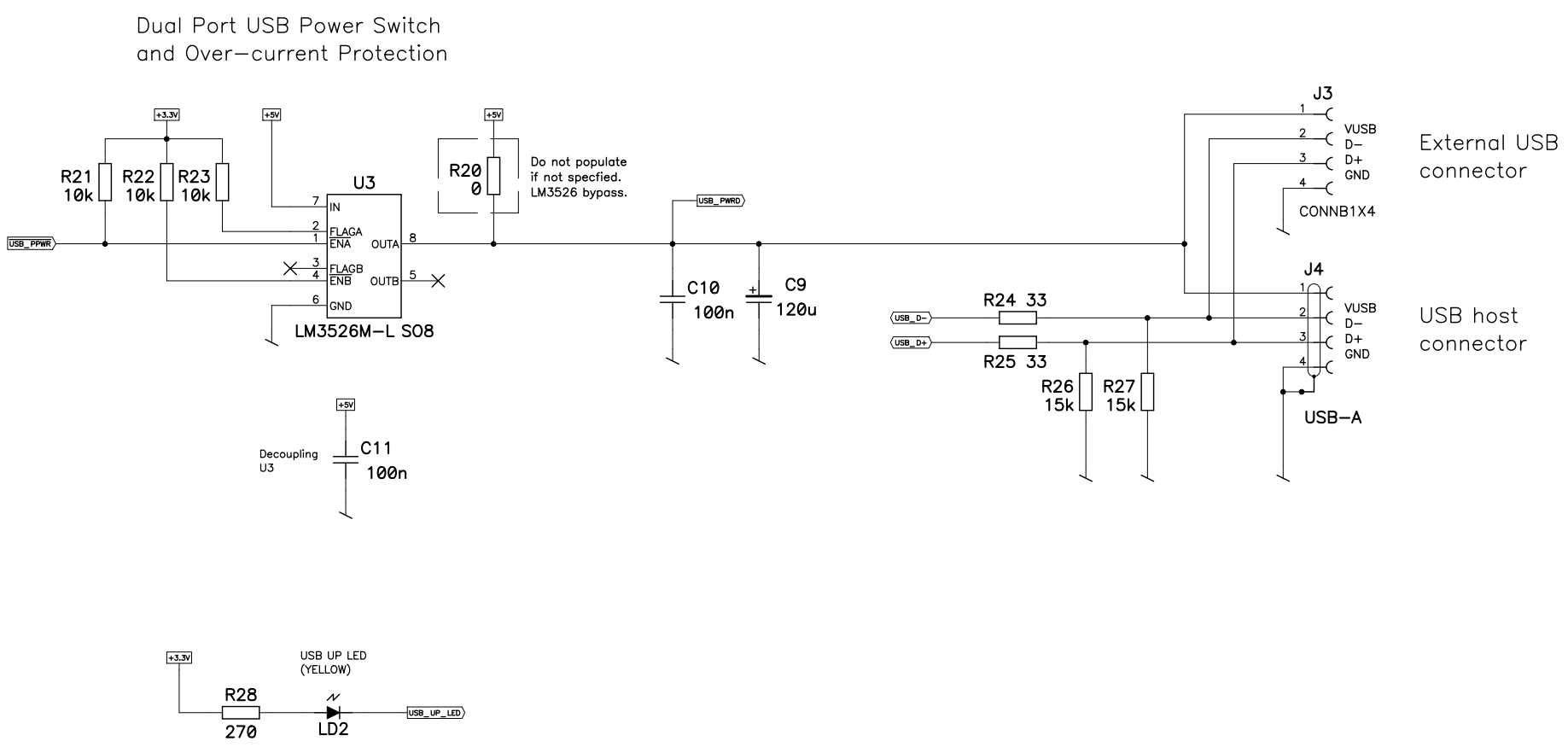






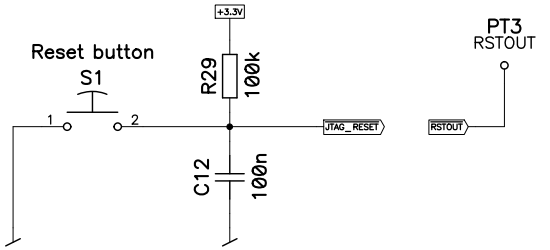
uGates – mini Android Accessory JTAG	DES	04.04.2012 metraic3
	REV	v0.1
HAUTE ECOLE VALAISANNE	2/6	P:\PCB\Student\TSTD\ uGates–miniV0_1_120412.sch

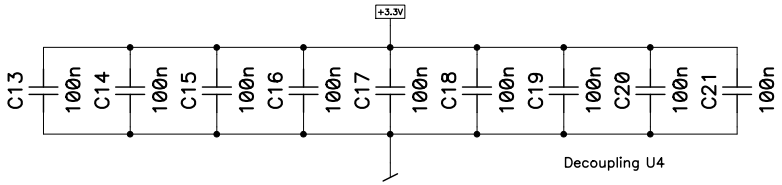
USB Host port

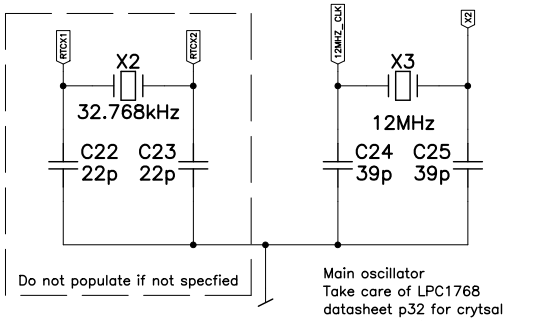


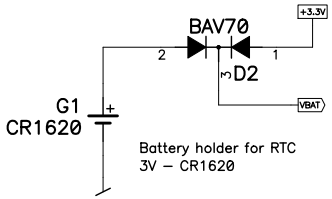
uGates – mini Android Accessory	Host	DES	04.04.2012 metraic3
		REV	v0.1
HAUTE ECOLE VALAISANNE		3/6	P:\PCB\Student\TSTD\ uGates-miniV0_1_120412.sch

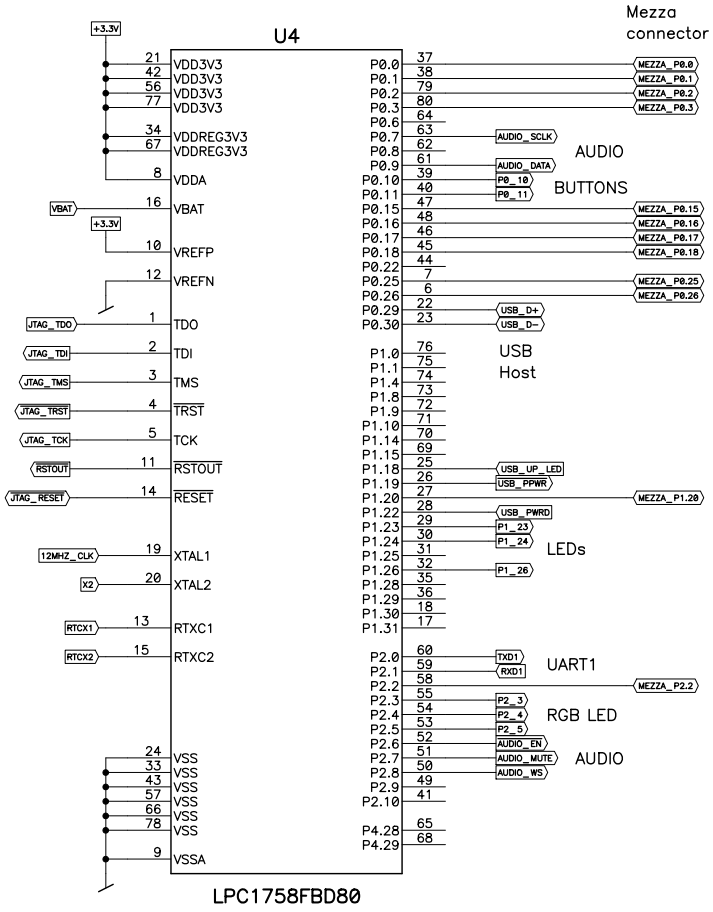
CPU – LPC1758





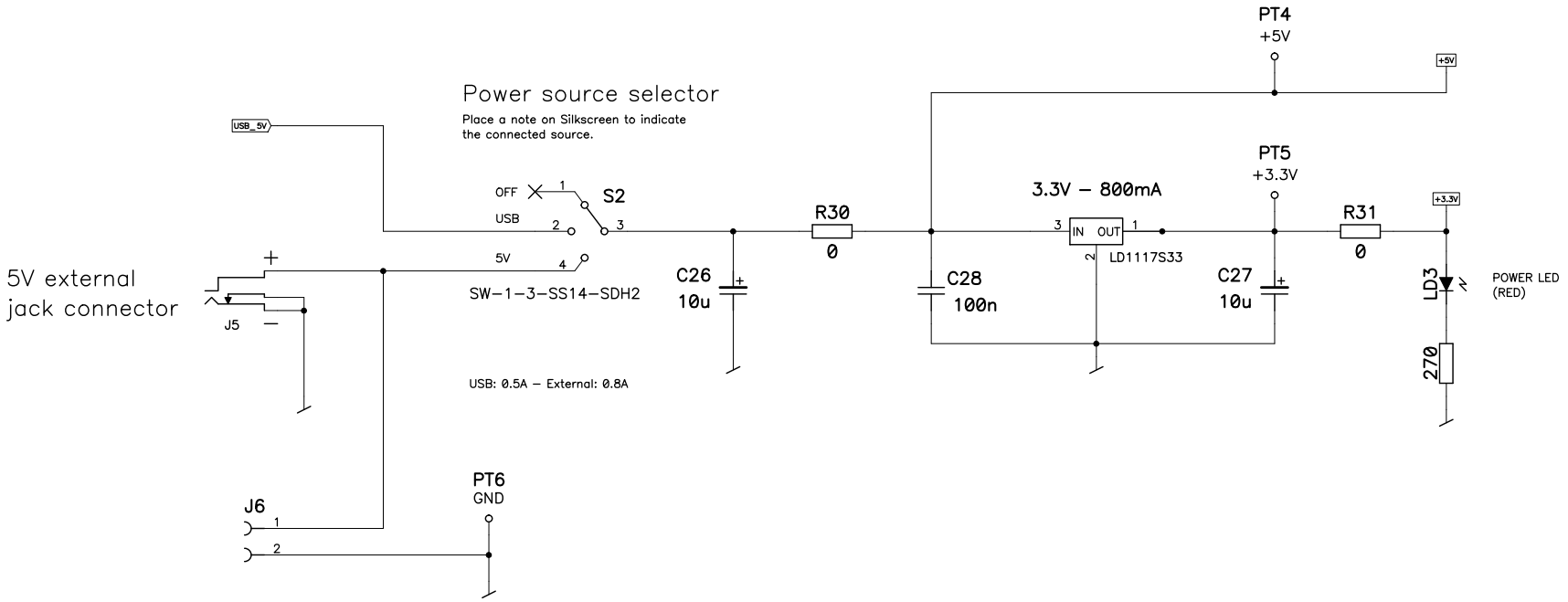






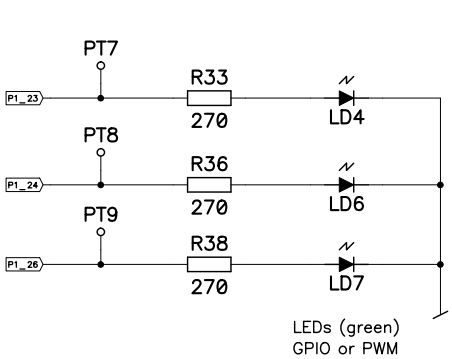
uGates – mini Android Accessory	CPU	DES	04.04.2012 metraic3
		REV	v0.1
HAUTE ECOLE VALAISANNE		4/6	P:\PCB\Student\TSTD\ uGates–miniV0_1_120412.sch

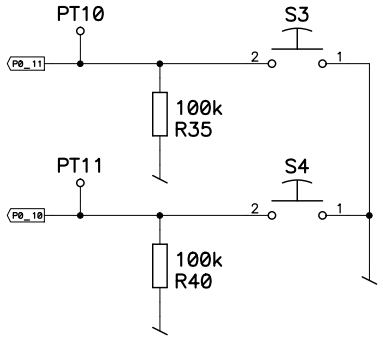
5V – 3.3V Power supply

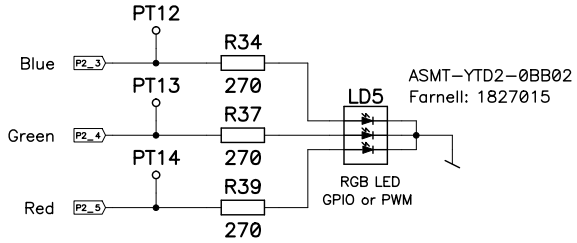


uGates – mini Android Accessory	Alim	DES	04.04.2012 metraic3
		REV	v0.1
HAUTE ECOLE VALAISANNE		5/6	P:\PCB\Student\TSTD\ uGates-miniV0_1_120412.sch

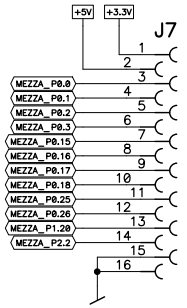
User LEDs & Buttons



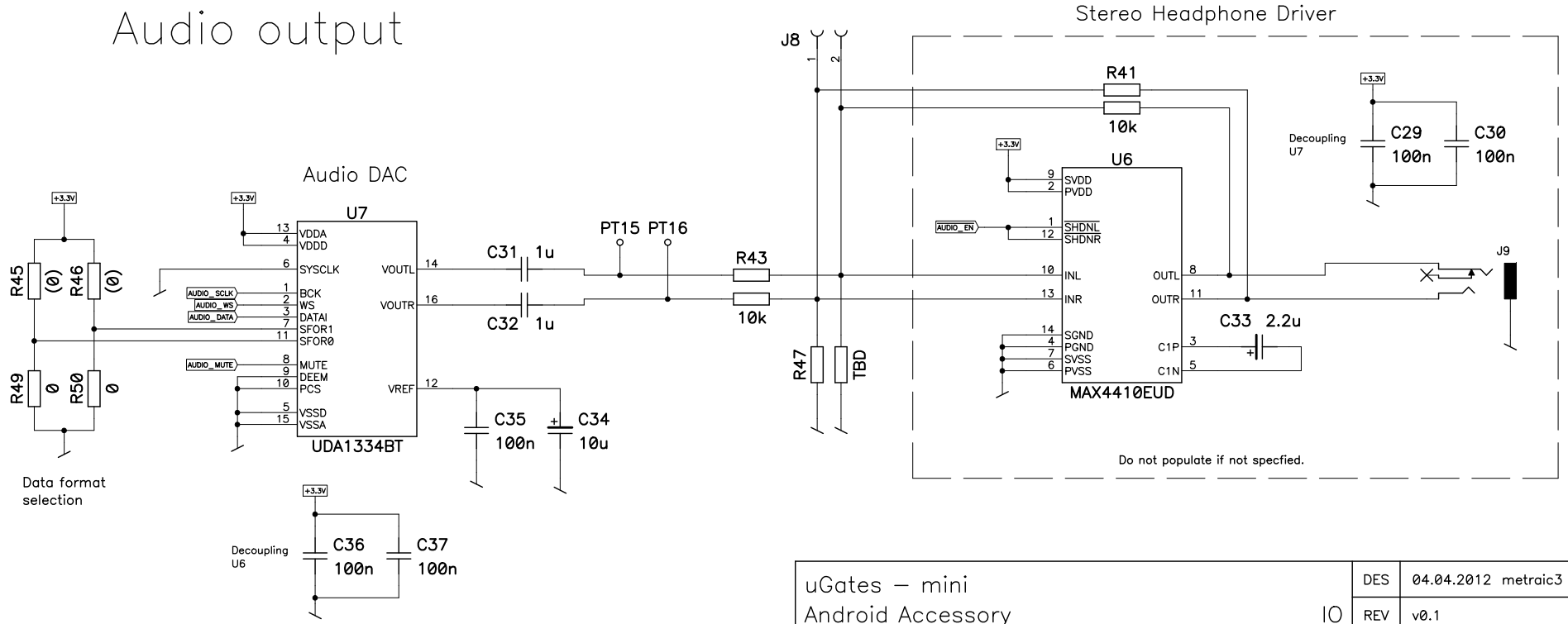




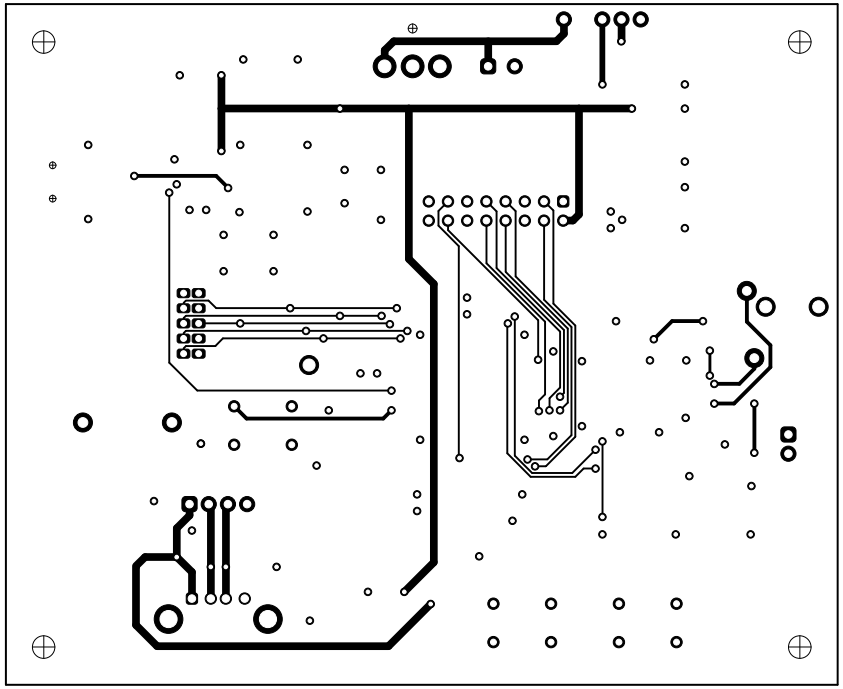
Mezza connector



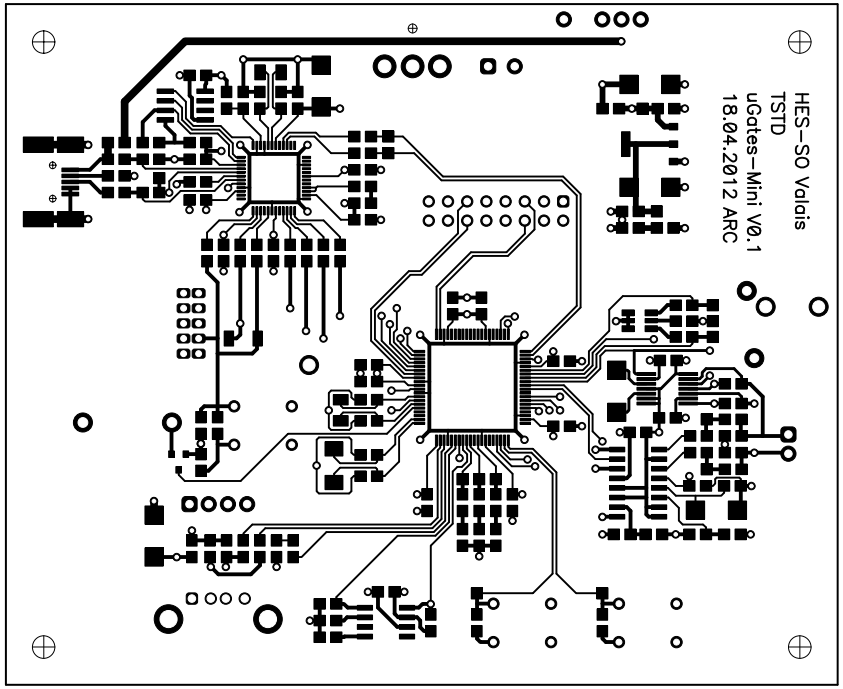
Audio output



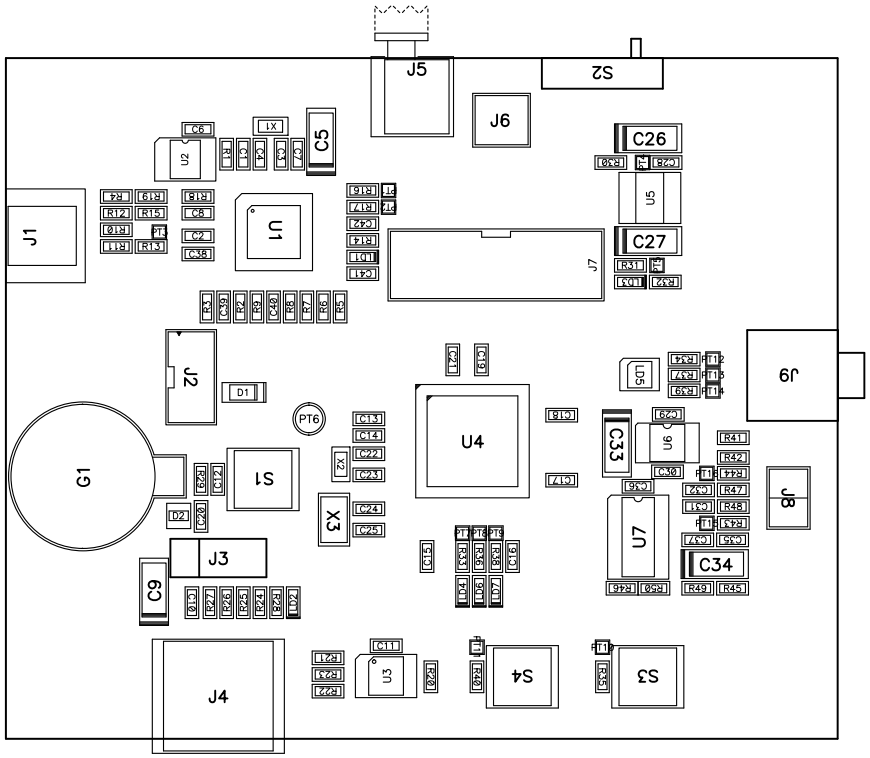
uGates – mini Android Accessory	IO	DES	04.04.2012 metraic3
		REV	v0.1
HAUTE ECOLE VALAISANNE		6/6	P:\PCB\Student\TSTD\ uGates-miniV0_1_120412.sch



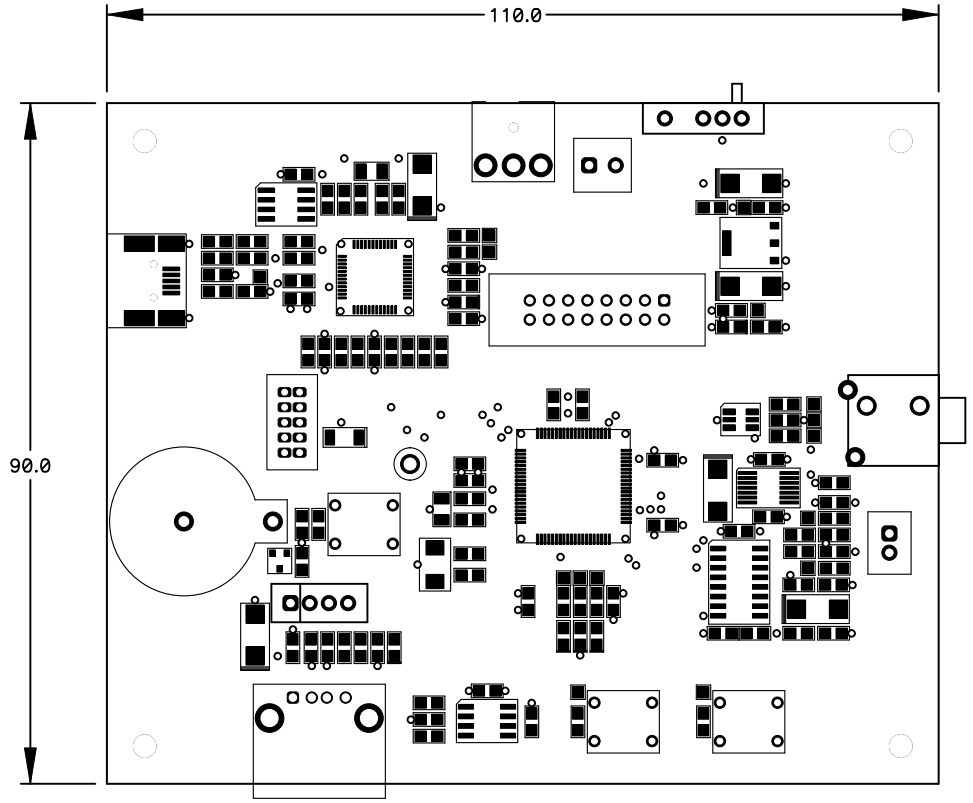
uGates – mini	DES	23.03.2012	
	REV	V0.1	
HAUTE ECOLE VALAISANNE	P:\PCB\Student\TSTD\ uGates–miniV0_1.pcb		



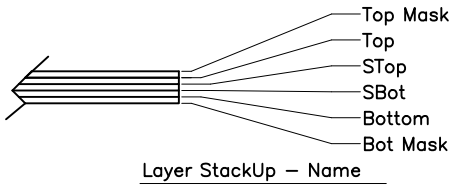
uGates – mini	DES	23.03.2012
	REV	V0.1
HAUTE ECOLE VALAISANNE	P:\PCB\Student\TSTD\ uGates-miniV0_1.pcb	



uGates – mini	DES	23.03.2012
	REV	V0.1
HAUTE ECOLE VALAISANNE	P:\PCB\Student\TSTD\uGates–miniV0_1.pcb	



Drill Table			
Hole Dia (mm)	Symbol	Quantity	Plated
0.500	+	115	Yes
0.800	X	22	Yes
0.900	Y	18	Yes
1.000	⊤	16	Yes
1.200	⧖	3	Yes
1.300	⋈	2	Yes
1.600	⊡	3	Yes
1.700	◈	3	Yes
2.300	⊠	2	Yes
3.000	◈	4	Yes



Layer StackUp – Name

uGates – mini	DES	23.03.2012
	REV	V0.1
HAUTE ECOLE VALAISANNE	P:\PCB\Student\TSTD\uGates–miniV0_1.pcb	

Commande 1

04.04.2012 **#CMD1**

Personne concernée:	Christopher Métrailler		
Mandat no.:	it/2012/23	Chef de projet / professeur:	Pierre-André Mudry
Salle:	A304		

Farnell

			Monnaie	CHF
Quantity	**Reference**	**Designation**	**Unit Price**	
3	9758232	EEPROM 1K 93LC46 SOIC8	0.40	1.19
3	1812370	CRYSTAL, 12MHZ, 18PF, SMD	2.10	6.30
3	1539380	CRYSTAL, 32.768KHZ SMD	5.10	15.30
3	1718545	LPC1758FBD80 CORTEX M3	12.55	37.65
3	1467779	LD1117S33CTR LDO 3.3V	1.45	4.35
3	1863245	LRTBG6SFV2BA4D5E - LED RGB	1.45	4.35
3	1854063	UDA1334 DAC Audio	3.25	9.75
3	1593370	MAX4410EUD Ampli audio	2.35	7.05
				85.94

Distrelec

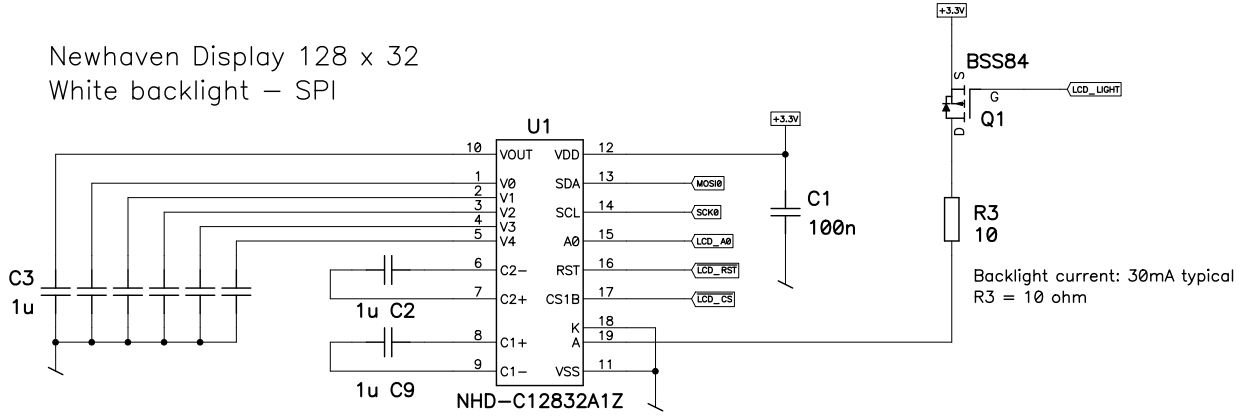
			Monnaie	CHF
Quantity	**Reference**	**Designation**	**Unit Price**	
3	200187	Commutateurs à coulisse	3.02	9.06
3	662202	LM3526M-L	2.48	7.44
3	973570	Porte-piles CR1620	3.13	9.39
3	970872	CR1620	1.62	4.86
3	662104	USB-UART/FIFO CMS FT2232D	14.15	42.45
				73.20

Annexe F

Dossier de fabrication de la carte d'extension.

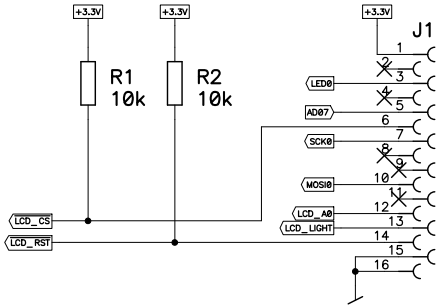
LCD

Newhaven Display 128 x 32
White backlight – SPI

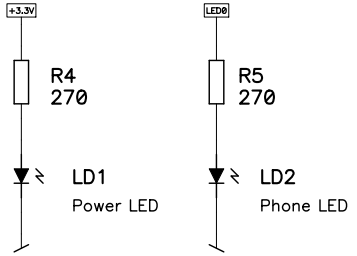


Recommended LCD connector: 1.5mm pitch pins, solder directly into PCB
Mouser : 763–C12832A1ZFSWFBW3

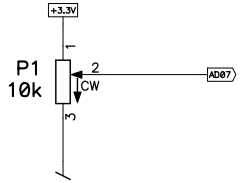
Mezza connector



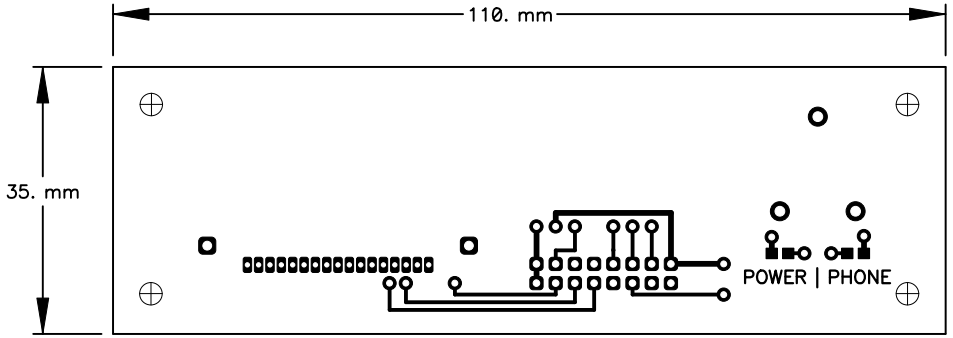
LEDs



Analog input

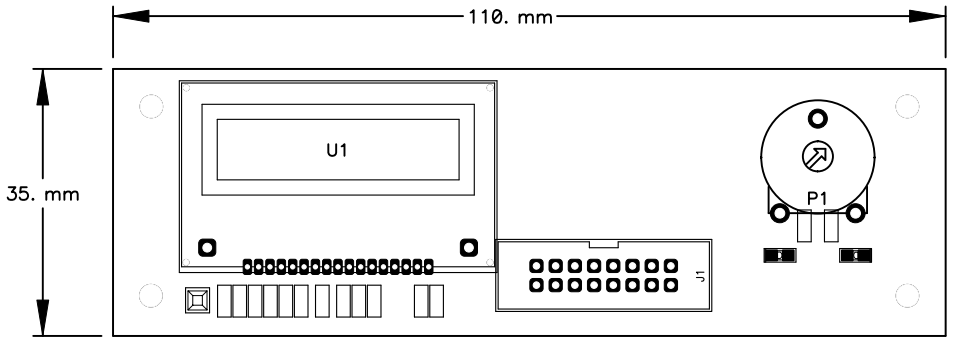


Mezza LCD Android Accessory	LCD	DES	12.06.2012 metraic3
		REV	V0.2
HAUTE ECOLE VALAISANNE		7/7	P:\PCB\Student\TSTD\ mezza–lcdV0_2_120612.sch



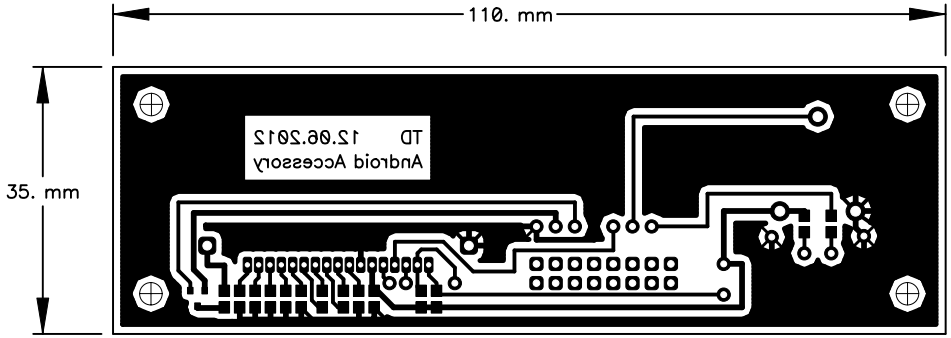
Drill Table			
Hole Dia (mm)	Symbol	Quantity	Plated
0.600	+	17	Yes
0.800	X	19	Yes
0.900	Y	16	Yes
1.200	⊤	2	Yes
1.300	⧖	3	Yes
3.000	⋈	4	Yes

Mezza LCD Android Accessory	DES	12.06.2012 metraic3
	REV	V0.2
HAUTE ECOLE VALAISANNE	P:\PCB\Student\TSTD\ mezza-lcdV0_2_120612.pcb	



Drill Table			
Hole Dia (mm)	Symbol	Quantity	Plated
0.600	+	17	Yes
0.800	X	19	Yes
0.900	Y	16	Yes
1.200	⊤	2	Yes
1.300	⧖	3	Yes
3.000	⋈	4	Yes

Mezza LCD Android Accessory	DES	12.06.2012 metraic3
	REV	V0.2
HAUTE ECOLE VALAISANNE	P:\PCB\Student\TSTD\ mezza-lcdV0_2_120612.pcb	



Drill Table			
Hole Dia (mm)	Symbol	Quantity	Plated
0.600	+	17	Yes
0.800	X	19	Yes
0.900	Y	16	Yes
1.200	⊤	2	Yes
1.300	⧖	3	Yes
3.000	⋈	4	Yes

Mezza LCD Android Accessory	DES	12.06.2012 metraic3
	REV	V0.2
HAUTE ECOLE VALAISANNE	P:\PCB\Student\TSTD\ mezza-lcdV0_2_120612.pcb	

Commande 2

23.05.2012 **#CMD2**

Personne concernée:	Christopher Métrailler		
Mandat no.:	it/2012/23	Chef de projet / professeur:	Pierre-André Mudry
Salle:	A304		

Mouser

Quantity	**Reference**	**Designation**	Monnaie **Unit Price**	CHF
4	763-C12832A1ZFSWFBW3	Ecran LCD blanc FSTN 128x32	10.52	42.08
6	771-UDA1334BTN2112	UDA1334BT/N2,112 SO16	1.48	8.88
6	520-ECS-600-18-10	Quartz 6MHz 18pF ECS-3X10	0.67	4.02
3	700-MAX4410EUD	Amplificateur audio MAX4410EUD+	1.35	4.05
				59.03

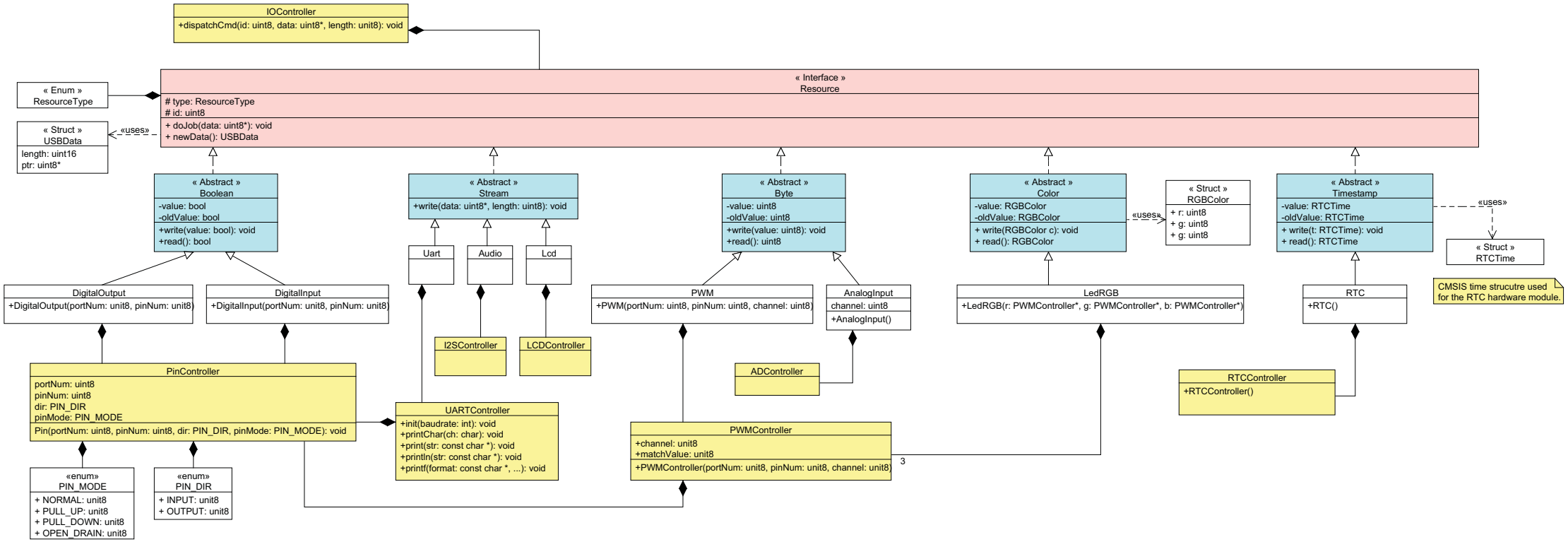
P-CAD Bill of Materials mezza-lcdV0_2_120612.pcb
uGATES - mini - mezza LCD

Count	ComponentName	RefDes	Value	Description
8	CAPS0805-MM	C2	1u	
		C3		
		C4		
		C5		
		C6		
		C7		
		C8		
		C9		
1	CAPS0805-MM	C1	100n	
1	CONNB2X8	J1		
1	FDN5618P	Q1	BSS84	
1	LCD-NHD-C12832A1Z	U1	NHD-C12832A1Z	
2	LED0805-MM	LD1	LST670-HK	Yellow LED for 3.3V supply
		LD2		Phone LED
1	RS0805-MM	R3	10	
2	RS0805-MM	R1	10k	
		R2		
2	RS0805-MM	R4	270	
		R5		
1	TRIM1T-PT15NV	P1	10k	

Annexe G

Diagramme de classes - accessoire USB.

Diagramme de classes - accessoire USB



Annexe H

Code de la classe *Resource*.

```cpp
// Resource C++ class
class Resource {
public:
  // Define all available types for resources
  enum ResourceType {
    BOOLEAN, STREAM, BYTE, COLOR, TIMESTAMP, UNKNOWN
  };

  typedef struct {
    uint16_t length; // length of data
    volatile uint8_t* ptr; // pointer to the byte buffer
  } USBData;

  Resource(uint8_t id, enum ResourceType t) {
    this->type = t;
    this->id = id;
  }

  ResourceType getType() {
    return type;
  }

  uint8_t getID() {
    return id;
  }

  // Pure virtual method. Must be implemented in other class.
  virtual void doJob(uint8_t *data) = 0;

  // Return null if the resource has nothing to send over USB
  virtual USBData newData() = 0;
protected:
  ResourceType type;  // resource type
  uint8_t id; // unique ID
};
```

Listing H.1 – Classe Resource C++

Annexe I

Spécification du protocole de l'application.

Application protocol

A frame always starts with a resource identifier. Frames are used for Data Access between the Android phone and the Accessory.

Resource ID	Data
RES_ID	n bytes
0x00	*0x01 ... 0x01 + n*

- **Boolean**
  Transmit a *boolean* value for a LED, a relay, a button or a switch for example.

Boolean value
0x00: false *others: true*
0x01

- **Byte**
  Transmit a byte value. It can be used for a PWM duty cycle for example.

Byte value
0x00 to 0xFF
0x01

- **Color**
  Transmit an RGB color on 24 bit, 8 bit for each color.

Red color	Green color	Blue color
0x00 to 0xFF	*0x00 to 0xFF*	*0x00 to 0xFF*
0x01	*0x02*	*0x03*

- **Stream**
  Transmit a *String* message with a variable length.

Length	String data
Data length	*String message with n char*
0x01	*0x02 ... 0x02 + n*

UART

Message
String message with n char
0x02 ... 0x02 + n

LCD

Screen position		Message
Pos X [0 .. 127]	*Pos Y [0 .. 3]*	*String message with n char*
0x02	*0x03*	*0x04 ... 0x04 + n*

Print a message on a fixed position on the screen.

AUDIO

Audio stream: signed PCM 16bit, mono, 16kHz, little indian
Audio resource ID = 128 (fixed).

Length		Audio PCM stream		
Number of samples		*Sample 0 LSB*	*Sample 0 MSB*	
0x01	*0x02*	*0x03*	*0x04*	*0x05 ... 0x05 + n*

- **Timestamp**
  Transmit a date and an hour. The timestamp is encoded as a String.

Hours	Minutes	Seconds	Day of month	Month	Year	
0 to 23	*0 to 59*	*0 to 59*	*1 to 31*	*1 to 12*	*MSB*	*LSB*
0x01	*0x02*	*0x03*	*0x04*	*0x05*	*0x06*	*0x07*

Example: 01.05.1990 - 18h30:22s Encoded: **18-30-22-01-05-19-90**

Annexe J

Thread de réception des données sur Android.

```
1  public void run() {
2    int ret = 0;
3    final byte[] buffer = new byte[16384];
4
5    while (ret >= 0) {
6      try {
7        // Read USB data
8        ret = mInputStream.read(buffer);
9      } catch (IOException e) {
10       break; // Nothing to read
11     }
12
13     // Valid command (minimal header length)
14     if (ret >= 2) {
15       // Extract the resource ID
16       final int resIndex = buffer[0];
17
18       // Run on UI thread to touch Views
19       runOnUiThread(new Runnable() {
20         public void run() {
21           if (mAccessory != null) {
22             AbstractResource r = mAccessory.getResource(resIndex);
23
24             if (r != null) {
25               // Send data to the corresponding resource
26               r.onNewUSBData(buffer);
27             }
28             else {
29               // Resource ID not valid
30               // Resource does not exist
31             }
32           }
33         }
34       });
35     }
36   }
37 }
```

Listing J.1 – Android audio Thread

Annexe K

Thread audio sur Android.

```
1  // Number of ns between each samples - 16.275kHz audio output
2  final int fSound = (int) ((1 / 16275.0) * 1000000000);
3
4  public void run() {
5    long lastTime = System.nanoTime();
6
7    while (true) {
8      // Compute the number of sample to send
9      long currentTime = System.nanoTime();
10     long deltaTime = currentTime - lastTime;
11     int sampleNbr = (int) ((deltaTime / fSound));
12
13     while (sampleNbr != 0) {
14       int read = sampleNbr * 2;
15       if (read > 2300)
16         read = 2300;  // Send max 2300 bytes = 1150 samples
17       sampleNbr -= read / 2;
18
19       // Create the audio frame
20       byte[] frame = new byte[3 + read];
21       frame[0] = (byte) 128;  // Indicate that's an audio frame
22       frame[1] = (byte) ((read >> 8) & 0xFF); // Number of audio samples
23       frame[2] = (byte) (read & 0xFF);
24
25       // Send the audio file
26       System.arraycopy(...);
27       mOutputStream.write(frame); // Send to Accessory
28       ...
29
30       // Thread delay - 30ms
31       try {
32         Thread.sleep(30);
33       } catch (InterruptedException e) {
34         ...
35       }
36
37       lastTime = currentTime;
38     }
39   }
40 }
```

Listing K.1 – Android audio Thread